Clube Delphi

Edição 167 :: R$ 14,90

DEVMEDIA

ISSN 1517990-7

Aplicações multi-idiomas
Internacionalize seus sistemas com suporte a vários idiomas

Bancos de dados para Business Intelligence
Converta uma base de dados transacional para Data Warehouse

Manipulação de PDF
Compactação, criação e extração de dados com GhostScript

DESENVOLVIMENTO WEB NO DELPHI

Vá além do desktop com sua linguagem preferida

MVP

R$ 1.000.000,00
INVESTIDOS EM CONTEÚDO NOS ÚLTIMOS 12 MESES.

APLIQUE ESSE INVESTIMENTO NA SUA CARREIRA...

E MOSTRE AO MERCADO QUANTO VOCÊ VALE!

CONFIRA TODO O MATERIAL QUE VOCÊ TERÁ ACESSO:

- + de **9.000** video-aulas
- + de **290** cursos online
- + de **13.000** artigos
- DEVMEDIA API's consumido + de **500.000** vezes

POR APENAS
R$ 69,90* mensais

*Tempo mínimo de assinatura: 12 meses.

ASSINATURA MVP MOST VALUABLE PROFESSIONAL

PRA QUEM QUER EXIGIR
MAIS DO MERCADO!

DEVMEDIA

Clube Delphi

167ª Edição 2016 ISSN 1517990-7 Impresso no Brasil

Editor Geral

Joel Rodrigues (joelrlneto@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Consultora Técnica

Daniella Costa (daniella.devmedia@gmail.com)

Capa e Diagramação

Romulo Araujo

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

www.devmedia.com.br/central
(21) 3382-5038

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site e para fechar parcerias ou ações específicas de marketing com a DevMedia, entre em contato com:

publicidade@devmedia.com.br

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

MVP

Assine agora e tenha acesso a todo o conteúdo da DevMedia: **www.devmedia.com.br/mvp**

DEVMEDIA

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site ClubeDelphi, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Joel Rodrigues - Editor da Revista
joelrlneto@gmail.com

Joel Rodrigues - **Editor Geral**

Editor da ClubeDelphi Magazine

Sumário

04 – Implementando aplicações multi-idiomas
[*Gutierry Antonio*]

Artigo no estilo Solução Completa

16 – Transformação de bases de dados transacionais em Data Warehouse
[*Jones Granatyr*]

Destaque - Reflexão

27 – Introdução ao desenvolvimento web no Delphi
[*Gutierry Antonio*]

41 – Manipulando PDF com GhostScript
[*Gutierry Antonio*]

Dê seu feedback sobre esta edição!

A ClubeDelphi tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/clubedelphi/feedback

Implemente aplicações multi-idiomas

Crie sistemas com suporte a vários idiomas

Devido ao caráter mundial da Web, internacionalização é uma característica cada vez mais buscada nas aplicações. Essa demanda tornou necessário o surgimento e amadurecimento contínuo de recursos de tradução que nos dias atuais facilitam, por exemplo, o acesso ao conhecimento por pessoas que não dominam outros idiomas além de sua língua materna. O Delphi, no entanto, cuja principal utilização é no desenvolvimento de aplicações desktop, ainda não possui soluções tão maduras.

O ITE (*Integrated Translation Environment*), presente no Delphi, permite criar aplicações multi-idiomas graças às suas funcionalidades de tradução, porém, esse ambiente não é muito favorável a aplicações de grande porte. Sua manutenção é consideravelmente complexa e, apesar dos arquivos de tradução não serem embutidos, a escolha de idiomas não é tão trivial para o usuário.

Há também no mercado alguns componentes e ferramentas desenvolvidas com esse propósito, como é o caso do componente TLang, que está disponível apenas para aplicações desenvolvidas com FireMonkey. Para a VCL (*Visual Component Library*) existem soluções semelhantes, porém, o preço de alguns desses componentes pode ultrapassar os US$ 800 para versões *Pro Edition*.

Neste artigo veremos uma forma mais eficaz e econômica para implementar essa funcionalidade nas aplicações. Serão explicadas as técnicas que podem ser utilizadas e como identificar a mais adequada para cada situação.

As abordagens exploradas neste artigo não se limitam à tradução entre idiomas. Elas podem ser usadas, por exemplo, para criar sistemas com mensagens customizáveis para cada ramo de empresas ou cliente, permitindo com que o usuário final defina os textos a serem utilizados na aplicação.

Nas próximas seções conheceremos alguns dos mecanismos que podem ser utilizados para implementar esse tipo de funcionalidade com o Delphi.

Fique por Dentro

Com grande frequência, as aplicações são desenvolvidas com suporte unicamente ao idioma nativo da equipe de programação, que insere toda a parte textual do sistema diretamente no design ou via código, sempre de forma estática. Desta forma, este software tende a ficar limitado ao seu país de origem ou, no mínimo, ter os clientes em potencial reduzidos, bem como sua rentabilidade e impacto no mercado. Como a indústria de software tem a capacidade de transcender facilmente os limites territoriais, oferecer suporte a vários idiomas pode aumentar consideravelmente as chances de sucesso da aplicação. Por isso, o tema apresentado neste artigo é útil para quem deseja aprender uma forma simples e flexível de alterar facilmente o idioma de suas aplicações Delphi.

ResourceString

ResourceString é um recurso que permite a criação de uma tabela de strings que funcionarão como constantes dentro da aplicação. Essa estrutura é muito utilizada internamente pelo Delphi na montagem de mensagens do IDE. Por exemplo, na unit *const* há exemplos de sua utilização. Os componentes também adotam esse método, evitando assim a utilização da declaração de um texto diretamente no código. Por exemplo, para a apresentação do termo "Open" em caixas de diálogo é utilizada a constante SOpenFileTitle.

Essa técnica facilita a manutenção e extensão do sistema, pois quando houver necessidade, não será preciso trocar o texto fixo em todos os lugares onde ele ocorre. Bastaria alterar seu valor na definição da constante e automaticamente essa mudança seria refletida em toda a aplicação.

Na **Listagem 1** vemos um exemplo da declaração de um ResourceString.

Um ResourceString pode ser declarado em qualquer lugar do projeto, ficando no mesmo contexto das declarações de constantes e variáveis. Para uma melhor organização e aproveitamento do código, no entanto, é uma boa prática fazer essa declaração em uma unit separada, deixando todos em um só local, ou separando por contexto/objetivo.

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 1. Declaração de ResourceString

```
01  ResorceString
02      sCaptionMinhaAplicacao = 'Meu Caption';
03      sCaptionBtnAbrir       = 'Abrir';
04      sMsgFecharAplicacao    = 'Deseja encerrar a aplicação';
```

Para acessarmos os valores definidos nos ResourceStrings basta referenciá-los pelos nomes que foram usados na sua declaração. Por exemplo, considerando as constantes da **Listagem 1**, poderíamos exibir a mensagem de confirmação de saída da aplicação da seguinte forma:

```
ShowMessage(sMsgFecharAplicacao);
```

Em todos os pontos do sistema onde for preciso confirmar a saída da aplicação não precisaremos repetir a mensagem na forma de string fixo, basta utilizar a constante *sMsgFecharAplicacao.* Além de evitar erros ortográficos nas demais mensagens, essa prática ajuda a manter a padronização dos textos, o que é também importante para oferecer uma boa experiência de uso.

Ao utilizar ResourceStrings, principalmente em arquivos externos, temos ainda os seguintes benefícios:

- Todos os programas que compõem a sua aplicação poderão utilizá-los sem que precisem ser reescritos;
- As mensagens podem ser formatadas para receberem parâmetros, aumentando ainda mais a padronização e estruturação do seu código.

Traduzindo a aplicação com ResourceString

Uma das formas de tradução utilizando o ResourceString é aplicada nas próprias mensagens do Delphi com a substituição dos arquivos de ResourceString originais. Porém isso pode ocasionar problemas ao atualizar a versão do IDE, pois será necessário alterar novamente esses arquivos. Além disso, pode ser que uma string não esteja mais na mesma unit que na versão anterior.

A outra forma é a criação de uma unit que altera os valores dos itens desejados para os novos. Esse será o conceito abordado nesse tópico, e para entender sua implementação crie um novo projeto e a partir do menu File > New > VCL Forms Application. No form principal, adicione um TButton e altere sua propriedade Caption para "Clique Aqui" e seu nome para *BtnSaudacao.*

Em seguida, crie no projeto uma pasta chamada *Idiomas* e a ela adicione uma nova unit chamada *dicionário,* onde deve ser implementado o ResourceString que pode ser visto na **Listagem 2**.

Retorne ao form principal e adicione a unit *dicionário* à seção *uses,* assim será possível acessar o ResourceString criado. Neste momento já podemos, acessar o item definido no ResourceString apenas referenciando-o pelo seu nome. Para testar, adicione ao evento OnClick do botão a seguinte linha:

```
ShowMessage(rsSaudacao);
```

Finalizadas essas etapas, já sabemos como utilizar o ResourceString em um projeto real. Façamos agora a parte da tradução propriamente dita. Para isso, crie uma unit chamada *idioma* dentro da pasta *Idiomas*. Crie uma classe nessa unit chamada *TIdioma* herdando de *TInterfacedPersistent*. Após esses passos a unit deve ficar como especificado na **Listagem 3**.

Listagem 2. Unit dicionario

```
01 unit dicionario;
02
03 interface
04
05 resourcestring
06  rsSaudacao = 'Ola Mundo';
07
08 implementation
09
10 end.
```

Listagem 3. Unit idioma

```
01 unit idioma;
02
03 interface
04 uses  Windows, classes;
05
06 type
07  TIdioma = class (TInterfacedPersistent)
08   protected
09    procedure AlterarResourceString(oResourceString: PResStringRec;
      psNovoValor: PChar);
10   public
11    Procedure Traduzir;virtual;Abstract;
12  end;
13
14 implementation
15
16 procedure TIdioma.AlterarResourceString(oResourceString: PResStringRec;
   psNovoValor: PChar);
17 var
18  POldProtect: DWORD;
19 begin
20  VirtualProtect(oResourceString,SizeOf(oResourceString^),
    PAGE_EXECUTE_READWRITE, @POldProtect);
21  oResourceString ^.Identifier := Integer(psNovoValor);
22  VirtualProtect(oResourceString,SizeOf(oResourceString^),
    POldProtect,@POldProtect);
23 end;
24 end.
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Na linha 9 está a declaração do método *AlterarResourceString* e na linha 16 a sua implementação. Esse método é responsável por alterar o valor atual do ResourceString para o desejado. Nesse caso estão sendo utilizados métodos internos para acessar o ponteiro da constante e alterar seu valor para o desejado. Na linha 11 está a declaração do método *Traduzir,* que será *virtual* e *abstract* para que seja implementado apenas nas classes filhas.

Após isso, adicione à pasta *Idiomas* as seguintes novas units:

- *es_ES* com a tradução para espanhol;
- *em_US* com a tradução para inglês;
- *uTraducao*, que será responsável por receber a solicitação de tradução e repassar para a classe correspondente ao idioma definido.

Ao fim dessa etapa o projeto deve ter uma estrutura semelhante ao que ilustra a **Figura 1.**

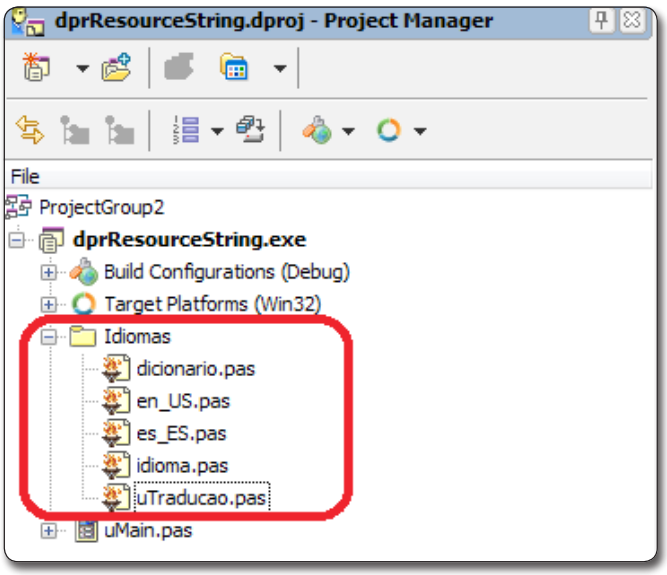
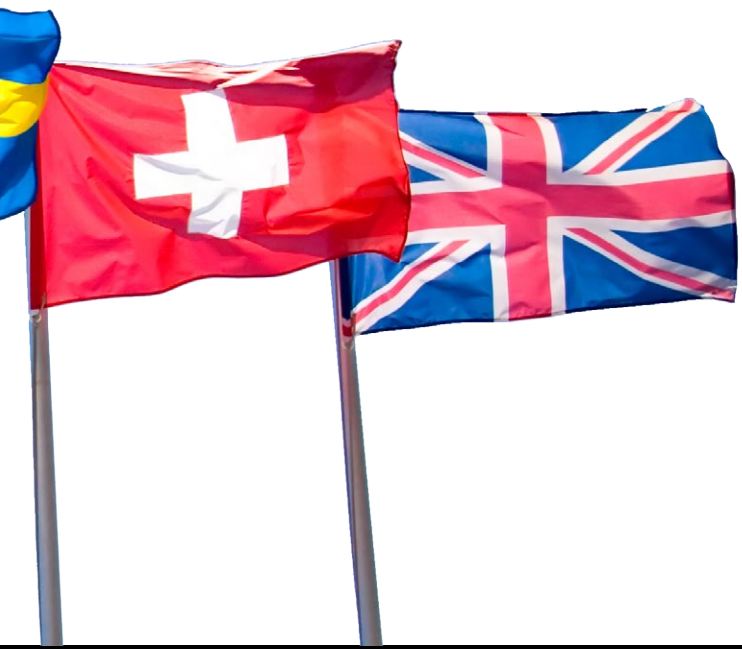


Figura 1. Estrutura do projeto de tradução por ResourceString



Agora, declare na *uTradução*, na cláusula *uses*, as seguintes units: *SysUtils, Rtti e idioma*. Nessa mesma unit crie a classe *TTraducao* e em seguida declare e implemente os métodos *Instanciar* e *Traduzir* de acordo com as **Listagens 4** e **5**.

Listagem 4. Método Instanciar da classe TTraducao

```
01 class function TTraducao.Instanciar(psClasse: String): TIdioma;
02 var
03   Context : TRttiContext;
04   classe  : TRttiInstanceType;
05 begin
06   classe := (Context.FindType(psClasse) as TRttiInstanceType);
07   result := TIdioma(classe.MetaclassType.Create);
08 end;
```

Listagem 5. Método Traduzir

```
01 class procedure TTraducao.Traduzir(psIdioma: String);
02 Var
03   oldioma : TIdioma;
04 begin
05   oldioma := Instanciar(psIdioma + '.T' + psIdioma);
06   try
07     oldioma.Traduzir;
08   finally
09     FreeAndNil(oldioma);
10   end;
11 end;
```

O método *Instanciar* é responsável por criar uma instância da classe correspondente ao pedido de tradução solicitado. Nesse método será feito o uso da RTTI (*Run-Time Type Information*) para buscar uma classe pelo seu nome. Na linha 6 a RTTI retorna a definição da classe através do seu nome, caso seja encontrada, e na linha 7 é passada para o result uma instância da classe correspondente.

No método *Traduzir* apenas é feita a chamada do *Instanciar*, passando como parâmetro o nome da unit mais o nome da classe. Depois da classe ser instanciada é chamado o método *Traduzir* da classe que alterará os valores do ResourceString para o idioma correspondente.

Na unit *es_ES* adicione na cláusula *uses* as units *classes*, *idioma* e dicionario. Crie a classe Tes_ES herdando de TIdioma e sobrescreva o método *Traduzir*. Na sua implementação adicione a seguinte linha:

```
AlterarResourceString(@rsSaudacao, 'hola mundo');.
```

Nesta linha estamos chamando o método *AlterarResourceString*, que foi definido na classe *TIdioma*, passando para ele o ponteiro da constante a ser alterada e o valor desejado.

Para que a classe seja encontrada pela RTTI ela deve ser registrada na seção *initialization* através do seguinte comando:

```
RegisterClass(Tes_ES);
```

Já na seção *finalization* deve-se remover o registro da classe, liberando assim os recursos utilizados durante sua utilização. Para isso, use o seguinte código:

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

```
UnRegisterClass(Tes_ES);
```

Para testar o funcionamento desse mecanismo adicione o seguinte comando no código do arquivo DPR do projeto, antes da chamada ao *Initialize*, assim o carregamento dos textos para tradução será feito antes da criação dos formulários:

```
TTraducao.Traduzir('es_ES');
```

Como a tradução foi feita já no início do carregamento do projeto, ao fazer referência aos valores definidos no ResourceString já teremos os textos traduzidos.

A unit *en_US*, bem como outras que se deseje, deve ser implementada da mesma forma que a *es_ES*, e ao utilizar o método *Traduzir* basta passar o nome da unit referente ao idioma desejado.

Seguindo a mesma ideia podemos definir os textos de todos os controles visuais do formulário. No entanto, é preciso ficar atento à necessidade de atribuir valores às propriedades em tempo de execução, pois só assim o valor utilizado será sempre o mais atual.

Essa técnica também permite o compartilhamento das units de tradução entre projetos, seja copiando a pasta ou compartilhando-a dentro do *Project Group*. É possível copiar a pasta e colá-la em todos os projetos, no entanto, ia requerer que sempre que uma alteração fosse feita em um local, fosse também replicada nos demais.

A desvantagem desse método é que os recursos ficam embutidos dentro da aplicação, dificultando a tradução por um usuário externo ou a liberação de um novo idioma sem uma nova compilação. Esse problema pode ser resolvido usando BPLs (Borland Package Library), que são módulos, permitindo a adoção de tecnologias sem refazer o sistema.

StringTable

O StringTable representa uma tabela de strings e é implementado na forma de recurso. Diferentemente do ResourceString, sua implementação se dá de forma externa através de um arquivo .res, e não em um .pas. Sua vantagem, se comparado ao método anterior, é que por estar contido em um arquivo externo ele pode ser carregado de forma dinâmica. Porém, os itens declarados na StringTable não podem ser acessados de forma tão "natural" como os do ResourceString, uma vez que o compilador ainda não tem conhecimento do conteúdo que pode ser encontrado dentro do arquivo de recurso. Uma outra desvantagem é que seus itens são numerados e sua manutenibilidade pode ficar complexa com o tempo, por isso não recomendamos essa técnica. Para retornar um item da StringTable deve-se fazer uso do método *LoadString*, passando como parâmetro o identificador da string na tabela. Na **Listagem 6** temos um exemplo da implementação desse recurso.

Seguindo o mesmo exemplo da **Listagem 1** foram definidas três strings. Nas primeiras linhas foi definido um conjunto de valores para posterior utilização dentro do bloco STRINGTABLE, o que serve para facilitar a identificação de cada string. Em seguida essas definições são utilizadas para atribuir os valores de cada item.

Listagem 6. Implementação de StringTable

```
01  #define sCaptionMinhaAplicacao   1
02  #define sCaptionBtnAbrir         2
03  #define sMsgFecharAplicacao      3
04  STRINGTABLE
05  BEGIN
06    sCaptionMinhaAplicacao , "Meu Caption"
07    sCaptionBtnAbrir      , "Abrir"
08    sMsgFecharAplicacao   , "Deseja encerrar a aplicação"
09  END
```

Um outro ponto importante a ser destacado é que para utilizar esse recurso no Delphi é preciso gerar os arquivos .res. Para isso deve-se utilizar o utilitário BRCC32.exe para gerar um recurso e em seguida adicionar à unit o comando {$R MeuRecurso.RES} abaixo de implementation. Isso fará com que ao compilar o programa, o recurso seja embutido no executável.

Para reduzir a repetição de código e facilitar a recuperação de strings definidas em StringTables podemos criar um método como o da **Listagem 7.** Nele declaramos um array de Char para receber o valor obtido e, recebendo por parâmetro apenas o identificador do item na tabela, repassamos para o método *LoadString* e retornamos o valor correspondente.

Listagem 7. Método para recuperar string em uma StringTable

```
01 function GetString(pID: Integer): string;
02 var
03    sReturn : array[0..255] of Char;
04 begin
05    LoadString(Hinstance, pID, sReturn,255);
06    Result := sReturn;
07 end;
```

Para utilizar esse método podemos proceder como na linha de código a seguir:

```
ShowMessage(GetString(1));
```

Apesar de simples, a utilização dessa técnica requer que sejam decorados os IDs das strings, o que pode ser um problema quando a tabela for muito grande, mas não é o objetivo desse artigo mostrar a solução.

Traduzindo a aplicação com StringTable

Para que fiquem mais claras as diferenças, vantagens e desvantagens do uso de StringTable quando comparado ao ResourceString, criaremos agora um exemplo seguindo os mesmos moldes do anterior. Realize então os seguintes passos do Delphi:

1. File > New > VCL Forms Application;
2. Crie uma nova unit e a nomeie como *uTraducao*;
3. Declare nessa unit a classe *TTraducao*, de acordo com a **Listagem 8**.

Essa classe tem a propriedade *idioma* que será utilizada para identificar em qual idioma uma palavra deverá ser recuperada.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Já o método *RetornarString* (**Listagem 9**), com parâmetro *pnIndex*, irá recuperar o valor da string no arquivo de recurso.

A implementação do *RetornarString* é bem semelhante ao método para recuperar a string em uma StringTable da **Listagem 7**. Na linha 4 tem uma nova variável, *nIndice*, que é definida na linha 6. Essa variável é definida com um prefixo, que é o valor do índice da propriedade *Idioma* mais o valor do índice que será localizado. Esse prefixo será para identificar qual o índice a ser obtido entre todos os recursos carregados no sistema.

Listagem 8. Classe TTraducao do exemplo com StringTable

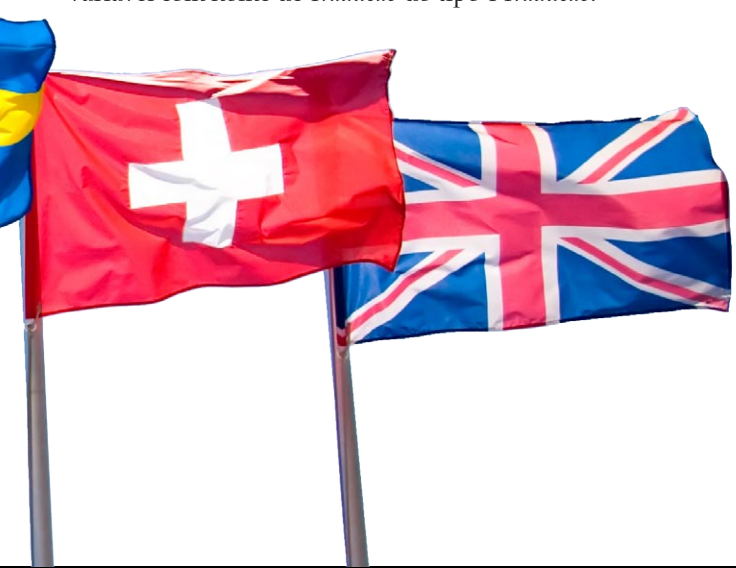
```
01 TTraducao = Class
02   private
03     fIdioma : Integer;
04   public
05     function RetornarString(pnIndex : Integer) : String;
06   published
07     property Idioma : Integer read fIdioma write fIdioma;
08 End;
```

Listagem 9. Implementação método RetornarString

```
01 function TTraducao.RetornarString(pnIndex : Integer) : String;
02 Var
03   sRetorno : array[0..255] of Char;
04   nIndice  : Integer;
05 begin
06   nIndice := (Idioma.ToString  + pnIndex.ToString).ToInteger;
07   LoadString(HInstance, nIndice ,sRetorno,SizeOf(sRetorno));
08   Result := sRetorno;
09 end;
```

Na linha 7 é utilizada a variável *nIndice* para recuperar o valor no recurso e no tamanho do buffer (último parâmetro): ao invés de preencher com o valor 255 fixo nó código é pego o tamanho da variável que receberá o retorno utilizando com o método *SizeOf*. Assim, ao alterar o tamanho do array de char de retorno não será necessário alterar o tamanho do buffer. No final, o result recebe o valor do *sRetorno*, assim como na implementação da **Listagem 7**.

Dando sequência, antes da seção *implementation* declare uma variável com nome de *Traducao* do tipo *TTraducao*.



Esse objeto será o responsável, nesse escopo, pela tradução e deve ser instanciado na seção *initialization* e liberado da memória na *finalization*, evitando assim o consumo de recursos desnecessários.

Para facilitar a manutenção do código e as trocas de idiomas, declare para cada idioma suportado uma constante e a defina com o valor que será o prefixo do índice na StringTable. Para este exemplo, adicione as seguintes constantes: *es_ES* com valor 1 e *en_US* com valor 2. Esses valores serão utilizados posteriormente para recuperar a classe específica do idioma que se deseja utilizar.

Em seguida, para cada idioma que se desejar traduzir crie um arquivo de extensão .rc dentro do diretório do projeto em uma pasta denominada *idiomas*. Esse projeto de exemplo deverá então conter um arquivo denominado de *es_ES.rc* e um segundo, *en_US.rc*. Estes conterão as tabelas com os identificadores e os valores dos textos a serem usados na tradução. Sua estrutura pode ser vista nas **Listagens 10** e **11**.

Listagem 10. Arquivo rc es_ES.rc (StringTable para tradução para espanhol)

```
01 #define stSaudacao          11
02 #define stCaptionBtnSaudacao   12
03 STRINGTABLE
04 BEGIN
05    stSaudacao , "hola mundo"
06    stCaptionBtnSaudacao , "haz click aquízir"
07 END
```

Listagem 11. Arquivo rc en_US.rc (StringTable para tradução para inglês)

```
01 #define stSaudacao  21
02 #define stCaptionBtnSaudacao  22
03 STRINGTABLE
04 BEGIN
05    stSaudacao   , "Hello Word"
06    stCaptionBtnSaudacao , "Click Here"
07 END
```

Note que os itens da StringTable de cada arquivo estão numerados de uma forma diferente. Além do valor sequencial, números são precedidos pelo mesmo identificador definido para cada idioma na unit *uTraducao*. Assim, os itens do arquivo de tradução para o espanhol estão numerados no formato 1+número (11, 12), enquanto os do inglês usam 2+número (21, 22). Isso é importante para facilitar a distinção entre os arquivos dos diferentes idiomas ao realizar a tradução.

Com os arquivos .rc criados, gere os .res a partir deles com o utilitário BRCC32.exe. Feito isso, copie os novos arquivos de recurso para o diretório do projeto e declare na unit *uTraducao*, abaixo de *implementation*, as diretivas {$R es_ES.RES} e {$R en_US.RES}, assim os resources serão importados na unidade.

Retorne então ao formulário principal e, na cláusula *uses*, referencie a unit *uTraducao*. Em seguida adicione ao form um TButton, cujas propriedades Caption e Name devem ser alteradas, respectivamente, para "Clique Aqui" e BtnSaudacao. No evento OnClick deve ser adicionado o comando a seguir, que basicamente exibe uma mensagem com o texto traduzido pela classe *TTraducao*:

```
ShowMessage(Traducao.RetornarString(1));
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Adicione ao evento OnCreate do form o seguinte comando, que atribui ao texto do botão o valor já traduzido:

```
BtnSaudacao.Caption := Traducao.RetornarString(2);
```

Agora, para definir o idioma que está sendo utilizado acesse o código do arquivo DPR e acima do *initialization* atribua o valor desejado à propriedade *Idioma* da classe *TTraducao*, como a seguir, e na sequência salve e execute o projeto:

```
Traducao.Idioma := es_ES;
```

Utilizar StringTable diminui o número de units no projeto e assim como no ResourceString, a tradução pode ser reutilizada para outros projetos, pois é implementada de forma isolada do código principal, tendo apenas que adicionar o recurso ao projeto. A desvantagem é que, assim como com o ResourceString, a tradução fica embutida dentro da aplicação, dificultando a tradução por qualquer pessoa externa ou a liberação de um novo idioma sem uma nova compilação. Esse problema também pode ser resolvido com a utilização de BPLs.

Um outro problema é que o desenvolvedor deve conhecer os números para as mensagens ou realizar uma consulta nas strings e seus índices. A solução para isso é criar uma unit com a lista de constantes com nomes claros para cada índice e utilizar as constantes nas chamadas e não os índices. Porém, para isso, em cada implementação nos arquivos de recursos deverá ser criada uma nova constante com o índice geral (valor sem o prefixo do idioma). Essa técnica dificulta a manutenção em mensagens (caption e texto) de formulários, pois todos os componentes e textos a serem traduzidos devem ser definidos ao inicializar o formulário.

ITE (Integrated Translation Environment)

Utilizaremos um terceiro mecanismo de tradução presente no Delphi de forma que possamos identificar suas principais características, vantagens e desvantagens com relação aos outros métodos.

Crie um novo projeto a partir do menu File > New > VCL Forms Application e ao form principal adicione um TButton. Altere a propriedade Caption do botão para "Clique Aqui" e seu nome para BtnSaudacao.

Em seguida declare no código do form um ResourceString da seguinte forma:

```
resourcestring
  rsSaudacao = 'Ola Mundo';
```

Para utilizar esse ResourceString podemos, por exemplo, adicionar a seguinte linha ao evento OnClick do botão e ver que será exibida uma mensagem com o conteúdo definido para a string:

```
ShowMessage(rsSaudacao);
```

A partir de agora trabalharemos no processo de tradução propriamente dito utilizando o ITE. Para isso acesse o menu Project > Languages > Add e será exibida uma mensagem solicitando que seja salvo o projeto. Em seguida aparecerá uma tela para selecionar os projetos a serem criados os arquivos de recurso de tradução. Clique em *Next*.

Na tela de "Add Languages" adicione os idiomas "Espanhol (Espanha - Tradicional) - ESP" e "Inglês (Estados Unidos) - ENU". Em seguida, clique em Next, Finish e Yes até finalizar o processo e assim faremos com que sejam adicionados mais dois projetos ao grupo, como mostra a **Figura 2**.

Para traduzir a unit basta que se clique duas vezes sobre sua correspondente no projeto secundário. Faça isso com a uMain.DFM do projeto ESP, por exemplo. Será aberto o "Translation Editor", como mostra a **Figura 3**. Esse editor apresentará todas as informações do DFM correspondente para que seja informada a tradução referente ao Espanhol. Para traduzir, basta alterar o valor da coluna "Espanhol (Espanha - Tradicional)", para que o valor da coluna "Status" será alterado de *Untranslated* para *Translated*. Em seguida localize o Caption do botão e altere para "Haga clic aquí".

Para alterar o valor do ResourceString ou de outro valor da aplicação, navegue ao aquivo .rc do projeto ESP, localize o valor *uMain_rsSaudacao* e altere para "Hola mundo". Faça o mesmo procedimento para o projeto em inglês, que possui a mesma estrutura. Em seguida clique com o botão direito sobre os projetos de inglês e espanhol e clique em "Make" para que os arquivos de recurso sejam carregados corretamente.

Para definir o idioma da aplicação, navegue até Project > Languages > Set Active com o projeto principal selecionado. Feita a escolha, compile o projeto e observe que o mesmo respeitou o idioma selecionado, estando agora com os textos traduzidos.

Ao compilar o projeto foram gerados dois arquivos de recursos ao lado do .exe, o *dprITE.ENU* e *dprITE.ESP*, correspondentes a

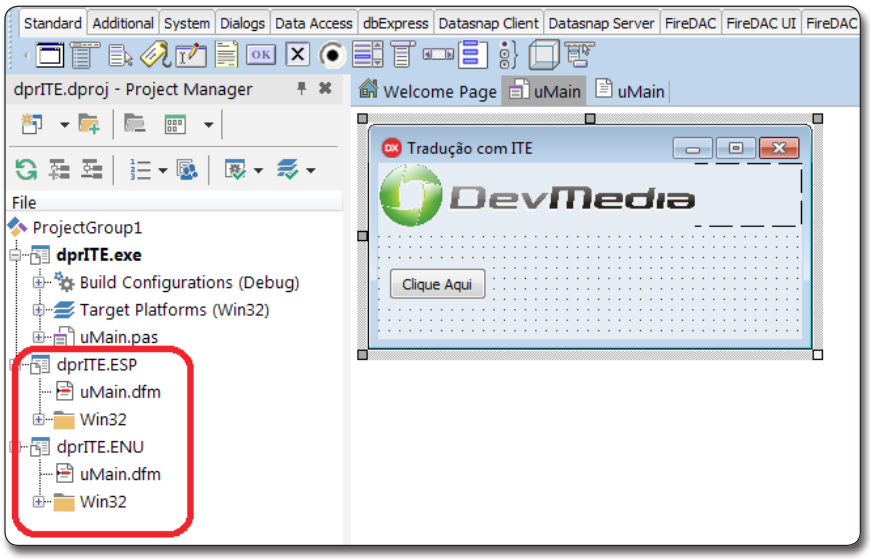


Figura 2. Projetos de tradução criados

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

cada idioma adicionado ao projeto. Esses arquivos devem ser distribuídos juntamente com a aplicação. Para definir outro idioma no computador do usuário deve ser alterado o valor do registro "**HKEY_CURRENT_USER\Software\Embarcadero\Locales**", onde o nome é o caminho completo da aplicação e o valor do registro deve ser correspondente ao idioma desejado.

A estrutura gerada pelo ITE, em partes, justifica o problema que este apresenta na tradução de projetos grandes. Primeiramente, é criado um projeto para cada idioma com todas as informações de seus DFM. Isso pode ficar complexo de dar manutenção e atrapalhar a abertura do projeto para projetos grandes devido ao tamanho dos subprojetos embutidos. Outro problema é que ao alterar um DFM e compilar um projeto com o idioma selecionado, pode-se notar em alguns casos que as alterações não refletem no .exe. Um exemplo seria a inclusão de um novo componente. Para que essas alterações surtam efeito deve-se utilizar o "Update Localized Projects", localizado em Project > Languages > Update Localized Projects e em seguida deve ser compilado o grupo de projeto. O ideal é que seja feito um build em todos os projetos, pois pular essa etapa pode gerar um .exe defeituoso.

Implementado um método de tradução próprio

Nesta seção veremos como implementar uma solução alternativa de tradução utilizando o melhor de cada método visto até aqui.

Utilizaremos aqui arquivos de forma completamente externa, permitindo ao usuário adicionar ou modificar um arquivo de tradução a qualquer momento no sistema. Isso dará maior flexibilidade à aplicação e viabilizará a distribuição da tradução de forma isolada do executável.

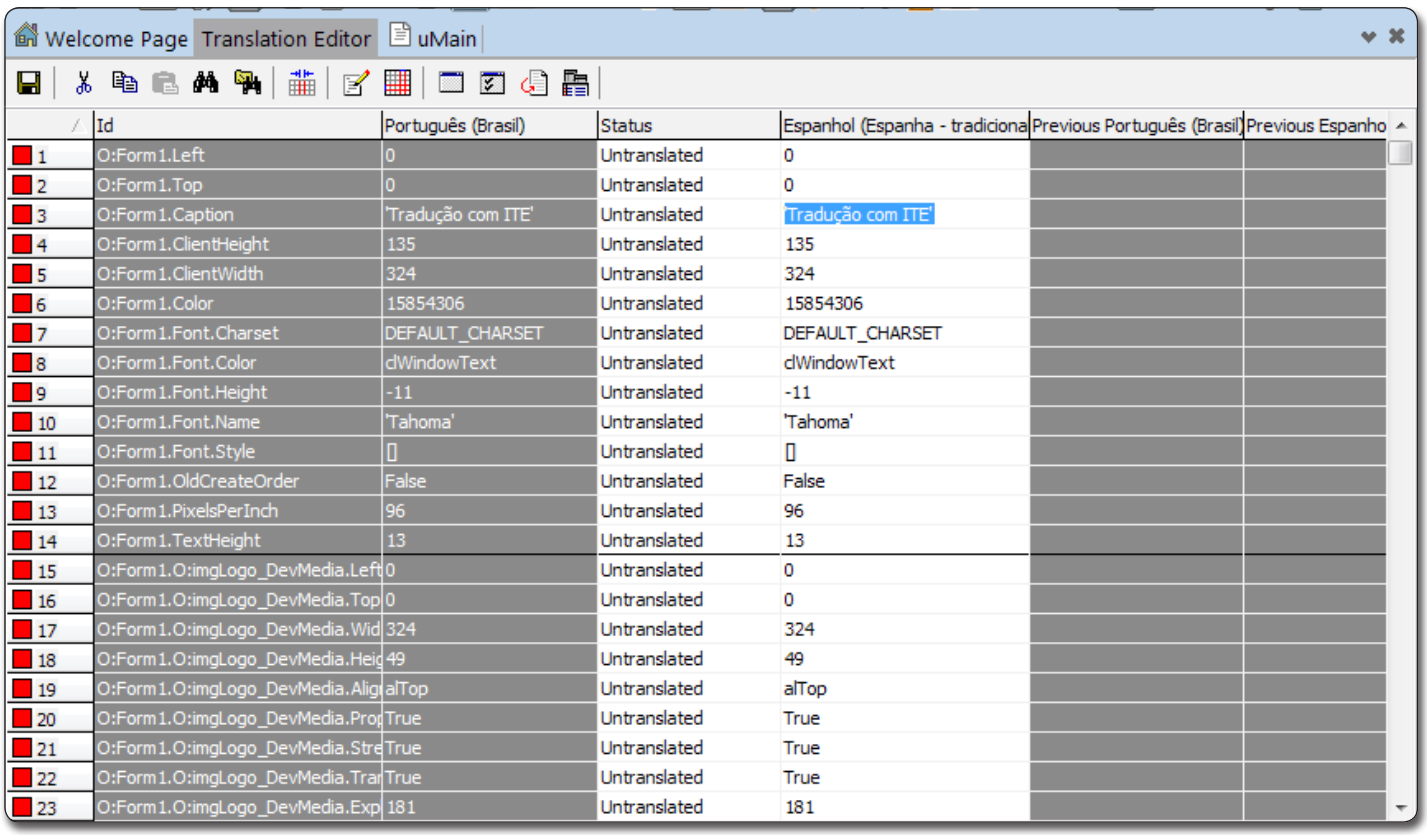
Será implementada uma rotina que irá percorrer no formulário os componentes visuais para realizar a tradução dos seus textos. Fazendo assim não haverá mais a necessidade de no evento OnCreate do formulário definir as traduções para cada componente.

Ao final será implementada uma rotina que permitirá alterar os valores dos ResourceStrings, para traduzir as mensagens padrões.

Crie um novo projeto pelo menu File > New > VCL Forms Application, adicione uma nova unit chamada *uTraducao* e nela crie a classe *TTraducao* de acordo com a **Listagem 12**.

Listagem 12. Classe TTraducao

```
01 TTraducao = Class
02   private
03     fIdioma : String;
04     ArqIng  : TIniFile;
05
06     procedure SetIdioma(const Value: String);
07   public
08
09     function RetornarString(psChave : String) : String;
10   published
11     property Idioma : String read fIdioma write SetIdioma;
12 End;
```



	Id	Português (Brasil)	Status	Espanhol (Espanha - tradiciona	Previous Português (Brasil)	Previous Espanho
1	O:Form1.Left	0	Untranslated	0		
2	O:Form1.Top	0	Untranslated	0		
3	O:Form1.Caption	'Tradução com ITE'	Untranslated	'Tradução com ITE'		
4	O:Form1.ClientHeight	135	Untranslated	135		
5	O:Form1.ClientWidth	324	Untranslated	324		
6	O:Form1.Color	15854306	Untranslated	15854306		
7	O:Form1.Font.Charset	DEFAULT_CHARSET	Untranslated	DEFAULT_CHARSET		
8	O:Form1.Font.Color	clWindowText	Untranslated	clWindowText		
9	O:Form1.Font.Height	-11	Untranslated	-11		
10	O:Form1.Font.Name	'Tahoma'	Untranslated	'Tahoma'		
11	O:Form1.Font.Style	[]	Untranslated	[]		
12	O:Form1.OldCreateOrder	False	Untranslated	False		
13	O:Form1.PixelsPerInch	96	Untranslated	96		
14	O:Form1.TextHeight	13	Untranslated	13		
15	O:Form1.O:imgLogo_DevMedia.Left	0	Untranslated	0		
16	O:Form1.O:imgLogo_DevMedia.Top	0	Untranslated	0		
17	O:Form1.O:imgLogo_DevMedia.Wid	324	Untranslated	324		
18	O:Form1.O:imgLogo_DevMedia.Heig	49	Untranslated	49		
19	O:Form1.O:imgLogo_DevMedia.Alig	alTop	Untranslated	alTop		
20	O:Form1.O:imgLogo_DevMedia.Pro	True	Untranslated	True		
21	O:Form1.O:imgLogo_DevMedia.Stre	True	Untranslated	True		
22	O:Form1.O:imgLogo_DevMedia.Tra	True	Untranslated	True		
23	O:Form1.O:imgLogo_DevMedia.Exp	181	Untranslated	181		

Figura 3. Translation Editor

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

O idioma será definido a partir da propriedade *Idioma*. A variável *Arqlng* da linha 4 será utilizada para carregar o arquivo INI com a lista das palavras traduzidas. Nesta primeira etapa implementaremos apenas os métodos *RetornarString* (que será responsável por localizar a tradução no INI) e o *SetIdioma*, que além de alterar o valor do field *fIdioma* também alterará o arquivo INI que será consumido pela variável *Arqlng*. Este arquivo também será utilizado pelo método *RetornarString* para recuperar o valor da string desejada.

No método *RetornarString* implemente a seguinte linha para retornar o valor localizado dentro da chave passada no parâmetro *psChave*, na seção *dicionario* de nosso arquivo INI:

```
Result := Arqlng.ReadString('dicionario',psChave, psChave);
```

Repare que como retorno default será utilizado o próprio valor do parâmetro.

Na sequência implemente o método *SetIdioma* de acordo com a **Listagem 13.** Esse método basicamente destrói a referência ao arquivo INI atual e carrega um novo na memória de acordo com o idioma recebido por parâmetro.

Listagem 13. Método SetIdioma

```
01 procedure TTraducao.SetIdioma(const Value: String);
02 begin
03   fIdioma := Value;
04
05   if Assigned(Arqlng) then
06    FreeAndNil(Arqlng);
07
08    Arqlng := TIniFile.Create(gsAppPath + Value + '.lng');
09   end;
```

Adicione no escopo global a variável *Traducao* do tipo *TTraducao* e a instancie na seção *initialization*. Em seguida, remova da memória essa referência na seção *finalization*, descartando os recursos utilizados.

Em seguida crie para cada idioma um arquivo INI no mesmo diretório da aplicação, nomeando-o como *es-ES.lng* para espanhol e *en-US.lng* para o inglês. O arquivo deve conter uma seção chamada *dicionario* e nela devem ser definidas as chaves e valores dos itens a serem traduzidos, como mostra a **Listagem 14**.

Para testar os métodos já criados retorne ao formulário principal e adicione na cláusula *uses* a unit *uTraducao*. Arraste para o form um TButton, altere o Caption para "Clique Aqui" e o nome para BtnSaudacao. Já no evento OnClick do botão adicione o seguinte comando, que exibe uma mensagem com o texto traduzido retornado pela classe *TTraducao*:

```
ShowMessage(Traducao.RetornarString('ola mundo'));
```

No código do arquivo DPR, acima de *Application.Initialization*, adicione o seguinte comando para definir o idioma antes de qualquer outra ação:

```
Traducao.Idioma := '<idioma>';
```

Em <idioma> podemos escolher 'es-ES' para espanhol ou 'en-US' para inglês.

Nesse momento a aplicação já é capaz de traduzir utilizando o método *RetornarString* da classe *Traducao*, mas para facilitar ainda mais o uso desse método, declare abaixo da variável global *Traducao*, na unit *uTraducao*, a seguinte função:

```
function _(psChave : String) : String;
```

Na implementação desse método adicione a seguinte linha:

```
result := Traducao.RetornarString(psChave);
```

Esse método apenas faz uma chamada ao *RetornarString* do objeto *Traducao*, porém com sintaxe simplificada. Assim, toda vez que for necessário chamar o método para tradução apenas utilizamos o "_" (underline). Com isso, a implementação do evento OnClick do botão de saudação pode ser alterado da seguinte forma:

```
ShowMessage(_('ola mundo'));
```

Com a tradução implementada será feita a implementação para traduzir os componentes visuais. Para isso, retorne à classe *TTraducao* e adicione na seção *private* a seguinte variável:

```
ListaPropriedades : TStringList;
```

Listagem 14. Arquivo INI de idioma

```
01 [dicionario]
02    la mundo=hola mundo
03  lique aqui=haz click aquízir
04 Tradução com método devmedia=Método de traducción con DevMedia
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Esta será utilizada para definir uma lista de quais propriedades de um componente serão traduzidas. Por exemplo, poderá ser definido que as propriedades Caption e Hint serão traduzidas.

Logo após, declare em *private* o seguinte método:

```
procedure Traduzir(poControl : TComponent; poRttiCtx : TRttiContext);Overload;
```

Este será responsável por percorrer os componentes do formulário e aplicar as traduções. Sua implementação deve ficar como está na **Listagem 15**. Na cláusula *uses* deve ser adicionada a unit *System.Rtti*.

Listagem 15. Método para tradução de componentes

```
01 procedure TTraducao.Traduzir(poControl: TComponent; poRttiCtx:
   TRttiContext);
02 var
03  i                : integer;
04  typeRtti         : TRttiType;
05  propRtti         : TRttiProperty;
06  sPropTraduzir    : String ;
07 begin
08  for sPropTraduzir in ListaPropriedades do
09  begin
10   typeRtti := poRttiCtx.GetType(poControl.ClassType);
11   propRtti := typeRtti.GetProperty(sPropTraduzir);
12   if Assigned(propRtti) then
13     propRtti.SetValue(poControl,RetornarString(
       propRtti.GetValue(poControl).ToString));
14  end;
15
16  for i:=0 to poControl.ComponentCount-1 do
17   Traduzir(poControl.Components[i],poRttiCtx);
18 end;
```

No primeiro loop é feito um *for each* onde, para cada item da *ListaPropriedades*, será verificado se o componente tem a propriedade desejada através do uso da RTTI. A partir disso aplica-se a tradução com a utilização do método *SetValue* para alterar o valor dessa propriedade a partir do que é retornado pelo método *RetornarString*. Depois de verificar todas as propriedades do componente, um segundo loop é aplicado, o qual percorre todos os componentes pertencentes ao componente atual e faz a chamada ao método de traduzir de forma recursiva.

Agora é necessário declarar o seguinte método como público:

```
procedure Traduzir(poControl : TComponent);Overload;
```

Este será o responsável por fazer a chamada do método tradução dentro da seção *private* e será o que deve ser utilizado no OnCreate e no OnShow dos formulários para aplicar a tradução aos seus componentes. A implementação do *Traduzir* deve ser feita como na **Listagem 16**, onde apenas cria-se o contexto de RTTI a ser trabalhado e o repassa como parâmetro para o método Traduzir *privado*.

Listagem 16. Método traduzir para tradução de componentes contexto publico

```
01 procedure TTraducao.Traduzir(poControl : TComponent);
02 var
03  ctxRtti : TRttiContext;
04 begin
05  ctxRtti  := TRttiContext.Create;
06  try
07    Traduzir(poControl,ctxRtti);
08  finally
09    ctxRtti.free;
10  end;
11 end;
```

Após essa etapa declare o evento OnCreate e nele instancie a *ListaPropriedades*, descartando-a da memória posteriormente no método OnDestroy. Já na seção *public* declare o método a seguir:

```
procedure AddPropriedadeParaTraducao(psPropriedade : String);
```

No corpo deste basta utilizar o método Add e adicionar à lista de propriedades a serem traduzidas o valor recebido por parâmetro:

```
ListaPropriedades.Add(psPropriedade);
```

A chamada a esse método deverá ser feita no evento OnCreate, onde as propriedades que devem ser traduzidas serão adicionadas à ListaPropriedades, como a seguir:

```
AddPropriedadeParaTraducao('Caption');
AddPropriedadeParaTraducao('Text'  );
```

Retorne em seguida ao formulário principal e no *FormCreate* adicione a seguinte linha para chamar o método de tradução passando como parâmetro o próprio formulário:

```
Traducao.Traduzir(Self);
```

Para definir o idioma a ser utilizado basta atribuir o valor desejado à propriedade *Idioma* do objeto *Traducao*.

Neste momento já poderá ser observado que os componentes foram devidamente traduzidos, inclusive o caption do formulário

Comparando com as técnicas utilizadas anteriormente, pode ser observado que a codificação é pequena.

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Programador Java: Por onde começar?

Descubra nesse vídeo como entrar na carreira Java com o pé direito!

DEVMEDIA

http://www.devmedia.com.br/programador-java-por-onde-comecar/33638

Não há a necessidade de criar um grupo de projetos, pois tem-se alto índice de reutilização em outros projetos. A vantagem de ter a tradução para cada idioma ser 100% isolada do projeto e não ficar embutida no exe, sem a necessidade de ter que implementar BPL para isolar a distribuição de pacotes de idiomas. Isso permite que qualquer um tenha acesso ao arquivo e faça suas traduções desejáveis ou adicione novos pacotes de idiomas. Não precisamos nos preocupar com IDs de strings e nem utilizar a palavra por extenso.

Para melhorar mais ainda essa implementação pode-se criar uma lista de ResourceStrings a serem traduzidas e fazer com que as mensagens padrões também sejam traduzidas. Para fazer isso, adicione na seção *private* a seguinte variável:

```
ListaResourceString : TList;
```

Em seguida, no OnCreate instancie o objeto da variável *ListaResourceString* e no OnDestroy destrua-o, limpando assim a memória.

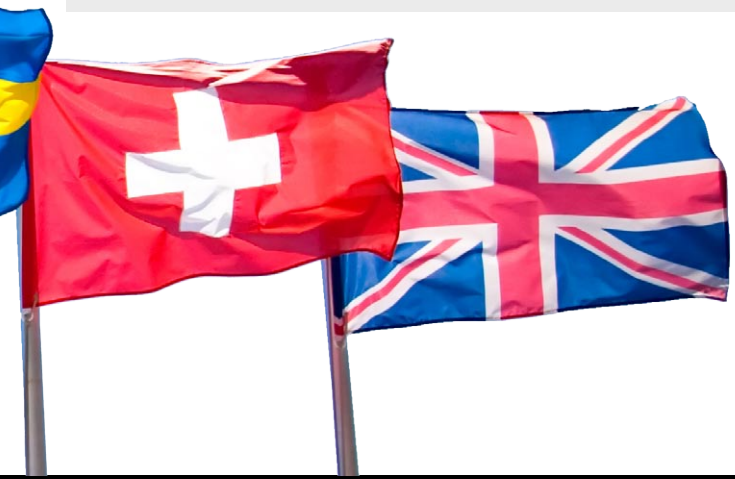
Logo em seguida, ainda no *private*, adicione o método *AlerarResourceString*, cuja implementação deve ser a mesma vista anteriormente:

```
procedure AlterarResourceString(poResourceString: PResStringRec; psNovoValor:
PChar);
```

O próximo passo é declarar o seguinte método, cuja implementação encontra-se na **Listagem 17**, abaixo da variável global *Traducao*:

Listagem 17. Método _pChar

```
01 function _pChar(psChave : String) : PWideChar;
02 Var
03   MaximumLength  : Integer;
04   str : String;
05   PWC : PWideChar;
06 Begin
07   str := _(psChave);
08   MaximumLength:= (Length(str)+1) * Sizeof(WideChar);
09   GetMem(PWC, MaximumLength);
10   StringToWideChar(str, PWC, MaximumLength);
11
12   Result := PWC;
13 End;
```



Esse método chama o _( ) e converte seu resultado em um PWideChar, retornando-o como resultado. Sempre que precisar de uma tradução em PWideChar poderá fazer a chamada da seguinte forma:

```
_PChar('Minha Mensagem');
```

Para concluir as declarações, adicione em *public* o método *TraduzirResourceStrings* e em sua implementação proceda como mostra na **Listagem 18**.

Listagem 18. Método TraduzirResourceStrings

```
01 procedure TTraducao.TraduzirResourceStrings;
02 Var
03  oResourceString: PResStringRec;
04 begin
05   for oResourceString in ListaResourceString do
06    AlterarResourceString(oResourceString,_pChar(
       LoadResString(oResourceString)));
07 end;
```

Neste método é feito um *for each* na *ListaResorceString* e para cada elemento é chamando o método *AlterarRecourceString*, passando como parâmetro o ResourceString a ser alterado e a tradução recuperada através do seu valor que foi retornado pelo método *LoadResString*.

Por fim, adicione o seguinte método, que registrará os ResourceStrings a serem traduzidos:

```
procedure AddResourceStringParaTraducao(poResourceString : PResStringRec);
```

Sua implementação é bem simples e consiste de apenas a seguinte linha:

```
ListaResourceString.Add(poResourceString);
```

Esse método deve ser invocado no evento OnCreate, onde deve-se adicionar os ResourceStrings a serem traduzidos, por exemplo:

```
AddResourceStringParaTraducao(@SMsgDlgYes);
```

No arquivo correspondente adicione o valor para a chave *&Yes*. Por exemplo, para o espanhol ficaria assim: &Yes=sí. Para testar, retorne ao formulário e adicione um MessageDlg, onde deve aparecer o texto já traduzido.

A lista a seguir mostra alguns ResourceStrings de mensagens padrões que podem ser adicionados ao processo de tradução:

- @SMsgDlgWarning,
- @SMsgDlgError
- @SMsgDlgInformation
- @SMsgDlgConfirm
- @SMsgDlgYes
- @SMsgDlgNo

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

- @SMsgDlgOK
- @SMsgDlgCancel
- @SMsgDlgHelp
- @SMsgDlgHelpHelp
- @SMsgDlgAbort
- @SMsgDlgRetry
- @SMsgDlgIgnore
- @SMsgDlgAll
- @SMsgDlgNoToAll
- @SMsgDlgYesToAll
- @SMsgDlgClose

Com essa técnica podemos não só traduzir mensagens adicionadas pelo usuário como mensagens já definidas no Delphi ou em componentes de terceiros. Basta informar o ResourceString desejado e adicionar sua string de tradução nos arquivos de idiomas.

Mantendo os arquivos de tradução separados e de fácil acesso, o desenvolvedor não precisa se preocupar com aspectos linguísticos desse processo, deixando isso a cargo de um especialista.

A distribuição de novos pacotes de idiomas também não depende de novos deploys de versão, e para facilitar ainda mais o trabalho do desenvolvimento, o método de tradução do formulário pode ser adicionado ao OnCreate do formulário pai, assim não será mais preciso efetuar a tradução em todos os formulários criados.

Para melhorar ainda mais essa implementação, a classe *TTraducao* pode ser transformada em um componente, facilitando seu reuso e distribuição.

Autor

Gutierry Antonio
gutierrydsn@hotmail.com

Entusiasta de Big Data e Data Warehouses, atua como Software Engineer e DBA MySQL. É graduado em Sistemas da Informação e Análise de Sistemas, e atualmente cursa especialização em Engenharia de Sistemas. Possui experiência em desenvolvimento com Delphi e Ruby on Rails com bancos de dados Firebird e MySQL, além de conhecimento de Java, C# e C++ com SQL Server e Oracle, bem como de AWS.

Links:

Embarcadero – Internacionalização de aplicações com Delphi
http://docwiki.embarcadero.com/RADStudio/Seattle/en/Creating_International_Applications_Index

GNU Gettext – GNU Gettext for Delphi and C++ Builder
http://dxgettext.po.dk

Helicon Translator - Helicon Translator for Delph
http://www.helicon.co.at/en/services-products/translator.html

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Transformação de bases de dados transacionais em Data Warehouse

Converta uma base de dados transacional em um Data Warehouse

O conceito de data warehouse (DW) está relacionado ao armazenamento de dados históricos de empresas em bases de dados, de forma que estes possam ser utilizados posteriormente para tomada de decisões estratégicas. Para que isso seja possível, após o DW estar construído, é necessária a utilização de técnicas de BI e/ou mineração de dados, com o intuito de encontrar conhecimento útil nos dados apresentados. A sua estrutura é a base para os sistemas de informações gerenciais e sistemas de apoio à decisão, pois ele possibilita a organização dos dados de forma a facilitar a integração com esses tipos de sistemas.

Dentro deste contexto, existem os sistemas OLTP (*Online Trasaction Processing*), que se enquadram na categoria dos sistemas de informações transacionais. Esse tipo tem a função principal de gerenciar as atividades diárias das empresas, como reserva de quartos em um hotel, processamento de folha de pagamento, controle de estoque, vendas em um supermercado, transações bancárias, dentre outros. Outras características dos sistemas transacionais são: possuir um alto grau de detalhe das informações, ser volátil e focado nas transações que estão ocorrendo em um determinado momento. Considerando o exemplo de um mercado, um sistema transacional pode ter a função principal de registrar as vendas, as quais são armazenadas em diversas tabelas que em geral seguem as formas normais da área de projeto de banco de dados.

Fique por Dentro

Data Warehouses têm a função de armazenar dados referentes às operações de uma determinada empresa, possibilitando a visualização de relatórios estratégicos, gráficos e/ou análises de grandes volumes de informações. É por meio de uma estrutura como esta que técnicas de BI (Business Intelligence) podem ser aplicadas, bem como a utilização de ferramentas OLAP (On-Line Analytical Processing) ou algoritmos de mineração de dados. Este artigo é útil para quem deseja efetuar a construção de uma estrutura assim a partir de uma base de dados transacional. Para isso, veremos neste artigo como desenvolver uma aplicação em Delphi para automatizar esse processo.

Por outro lado, um data warehouse deve ser estruturado para possibilitar um alto nível de agregação das informações, bem como possibilitar uma visão dos dados integrados referentes às atividades ocorridas durante um tempo pré-determinado. Enquanto um sistema OLTP é volátil e rápido para oferecer detalhes minuciosos nos dados, um DW deve ser projetado para ser estável e rápido para fornecer agregações sobre os dados. A **Figura 1** apresenta um comparativo que mostram as diferenças entre ele e um sistema transacional, na qual é possível observar que as principais operações em OLTP são: incluir, alterar, excluir e acessar. Por outro lado, em um data warehouse os dados são carregados e o foco principal é o acesso. Em geral, os dados armazenados em uma base desse tipo não podem ser alterados depois de carregados a menos que seja necessário fazer algum tipo de correção.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

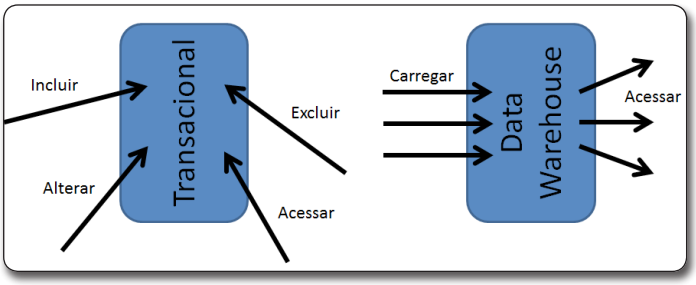


Figura 1. Sistema transacional x data warehouse

Para que a construção de um data warehouse seja possível é necessário existir pelo menos uma fonte da qual os dados serão extraídos. Normalmente, esses dados são importados dos sistemas transacionais/operacionais que existem nas empresas, porém, diversas outras fontes podem ser utilizadas, tais como dados de webservices, documentos de texto ou planilhas eletrônicas. A **Figura 2** apresenta um esquema que mostra esse fato, no qual é possível visualizar diversas fontes de dados unindo-se ao data warehouse no centro da figura. Após a junção dos dados, diversos tipos de acesso podem ser feitos, desde a utilização de geradores de relatórios tradicionais (Fast Reports, por exemplo), ferramentas OLAP (FastCube, por exemplo) e/ou outros aplicativos.

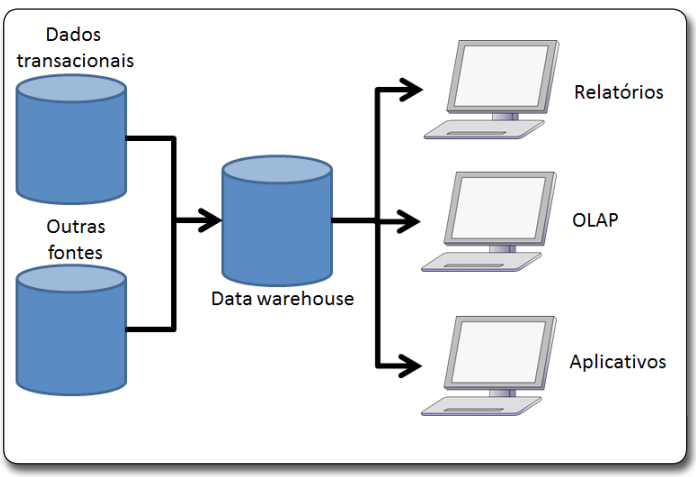


Figura 2. Fontes de dados para o data warehouse

Esse tipo de base de dados possui características distintas de uma base transacional. Por esse motivo, caso uma empresa deseje trabalhar com essa tecnologia, primeiramente precisa que sua base transacional, que utiliza normalização de dados, seja convertida para um novo padrão que se adeque às características do data warehouse, chamado de modelagem multidimensional. Esse processo normalmente não é trivial, pois requer diversas adaptações no novo banco, bem como a importação dos dados transacionais para o novo formato. Baseado nisso, o objetivo do presente artigo é DW. Será construído um importador no Delphi 10 Seattle utilizando o SQL Server e os componentes da biblioteca FireDAC. Em suma, o objetivo principal é transferir os dados, abordando os principais processos envolvidos nesta tarefa, bem como a redução de registros e o pré-processamento dos dados existentes.

As próximas seções apresentam as duas bases de dados utilizadas no exemplo proposto, bem como explicações sobre as principais diferenças entre uma base de dados transacional e um DW.

Entendendo a base de dados transacional

A **Figura 3** apresenta o diagrama entidade relacionamento da base transacional. O objetivo principal desta base de dados é controlar a venda de produtos nas diversas filiais de uma determinada empresa, existindo para isso nove tabelas. É possível notar que o controle dos produtos é feito por quatro delas, a saber: *produtos*, *produto_categorias*, *produto_departamentos* e *produto_familias*. A tabela *produtos* armazena todos os itens existentes na loja, sendo representada somente pelo nome e pela categoria do produto. A categorização dos produtos é dada pelas três outras tabelas. Para exemplificar, considere a família de produtos *Bebidas*, relacionado a ela o departamento *Bebidas alcóolicas* e, por fim, o relacionamento com a categoria *Cervejas*. Em suma, a cerveja pertence ao grupo das bebidas alcoólicas, que por sua vez está ligado à família de produtos das bebidas.

A tabela *cidades* está relacionada com as tabelas de *clientes* e *lojas*, que por sua vez estão ligadas com a tabela de *vendas*. Como pode ser notado no esquema da base de dados, cada venda possui um cliente e uma loja na qual a venda foi realizada. Uma venda pode conter vários produtos, os quais são representados pela tabela *vendas_produto*.

Nota

O esquema apresentado na **Figura 3** tem o intuito de ser didático apenas para mostrar a transformação de uma base transacional em um data warehouse. Por esse motivo, as tabelas apresentam um número reduzido de campos.

Entendendo o data warehouse

A **Figura 4** apresenta o diagrama entidade relacionamento da estrutura do DW já definida. É possível observar que a quantidade de tabelas diminuiu de nove para cinco, apresentando somente os dados que serão utilizados nas análises. Foram adicionados os campos *cidade* e *estado* diretamente nas tabelas de *clientes* e *lojas*. Nessas tabelas, pode-se notar também que os campos *nome* foram retirados, pois este tipo de informação normalmente não é relevante em agrupamento e/ou análise de dados. Utilizar o nome do cliente não traria muitos benefícios e *insights* para os gestores, pois talvez não faça muito sentido agrupar as vendas de todos os clientes com o nome de Maria, por exemplo. Já com relação às lojas, talvez fosse interessante agrupar os totais de vendas pelo nome da loja, pois assim o gestor poderia tomar decisões baseado em cada filial. Neste data warehouse, então, este campo não foi considerado.

Outra abreviação realizada na base de dados foi a junção das quatro tabelas sobre os produtos somente em uma, a qual possui a família, o departamento e a categoria dos produtos (*produtos_classe*). Essa redução foi necessária porque na tabela de produtos existem 10.281 registros distintos, o que tornaria o processo de

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

análise e agrupamento dos dados inviável. Desta forma, foram utilizadas somente as 47 categorias existentes na base transacional, reduzindo muito a quantidade de registros únicos a serem processados; o que facilita também o agrupamento dos dados. Com isso, é possível visualizar os valores totais vendidos em cada grupo, como o total de unidades vendidas na categoria *Cervejas*.

A tabela *vendas_fato* é a principal, pois apresenta os campos que são as medidas, ou seja, as unidades vendidas e o valor vendido. Essa tabela, além de apresentar as chaves estrangeiras para clientes, lojas e classe de produtos, armazena os valores totais de cada venda, os quais são somados e agrupados da tabela *vendas_produto* do esquema transacional apresentado na **Figura 3**. É possível notar também a tabela *tempo*, a qual não consta no modelo transacional e tem o intuito de realizar o agrupamento dos dados pelo dia da semana, mês e quadrimestre. Esses dados podem ser obtidos por meio do campo *data_venda* da tabela *vendas* da base transacional. Neste contexto, a utilização de tabelas de tempo neste formato é bastante comum em cenários de *Business Intelligence*, já que essa separação da data completa em campos individuais possibilita melhor desempenho das consultas e melhor poder para resumo dos dados.

Essas alterações na base de dados podem parecer, à primeira vista, redundâncias e divergências com as formas normais se comparados ao modelo transacional. Porém, neste tipo de aplicação essas redundâncias são permitidas, pois os dados armazenados no data warehouse são de cunho histórico e utilizados somente para consultas. O usuário final de um sistema de informação transacional não terá acesso a esses dados, e operações rotineiras de inclusão, alteração e exclusão não são usualmente permitidas após as tabelas do DW serem preenchidas.

Por esse motivo, não há a obrigatoriedade de manter integridade referencial na base de dados, pois como os registros não são excluídos, a integridade referencial não será quebrada em um relacionamento entre chave primária e chave estrangeira, por exemplo. Outro ponto a frisar é que como sua característica é possibilitar consultas, a redução do número de *joins* poderá reduzir os tempos de consultas à base de dados, dependendo da quantidade de tabelas e registros. Um exemplo claro na **Figura 4** é a tabela *produtos_classe* que contém os principais dados dos produtos em uma única tabela, evitando consultas SQL com junção em quatro tabelas, originalmente oriundas do modelo transacional apresentado na **Figura 3**.

A estrutura apresentada na **Figura 4** é chamada de modelo multidimensional e, neste contexto, é importante frisar três conceitos: tabela fato, medidas e dimensões. A tabela fato é onde encontram-se os valores numéricos que irão representar algum tipo de totalização, como os campos *unidades_vendidas* e *valor_vendido* da tabela *vendas_fato*. Esses campos é que são agrupados e visualizados pelas ferramentas OLAP de *Businness Intelligence*, os quais são conhecidos como medidas. Por fim, as dimensões são tabelas que possibilitam que as medidas sejam visualizadas de forma a possibilitar o gestor tomar decisões e/ou analisar os dados existentes na empresa. Por exemplo, com base na **Figura 4** é possível agrupar os dados pela família dos produtos, pelo tipo da loja ou pelo dia da semana em que as vendas foram realizadas.

Existem basicamente dois tipos de modelos dimensionais: estrela (*star scheme*) e floco de neve (*snow flake*). No primeiro, as tabelas que representam as dimensões estão todas ligadas à tabela fato, o que o deixa mais simples e possibilita melhores formas de navegação nos dados pelas ferramentas OLAP, como o FastCube, por exemplo.

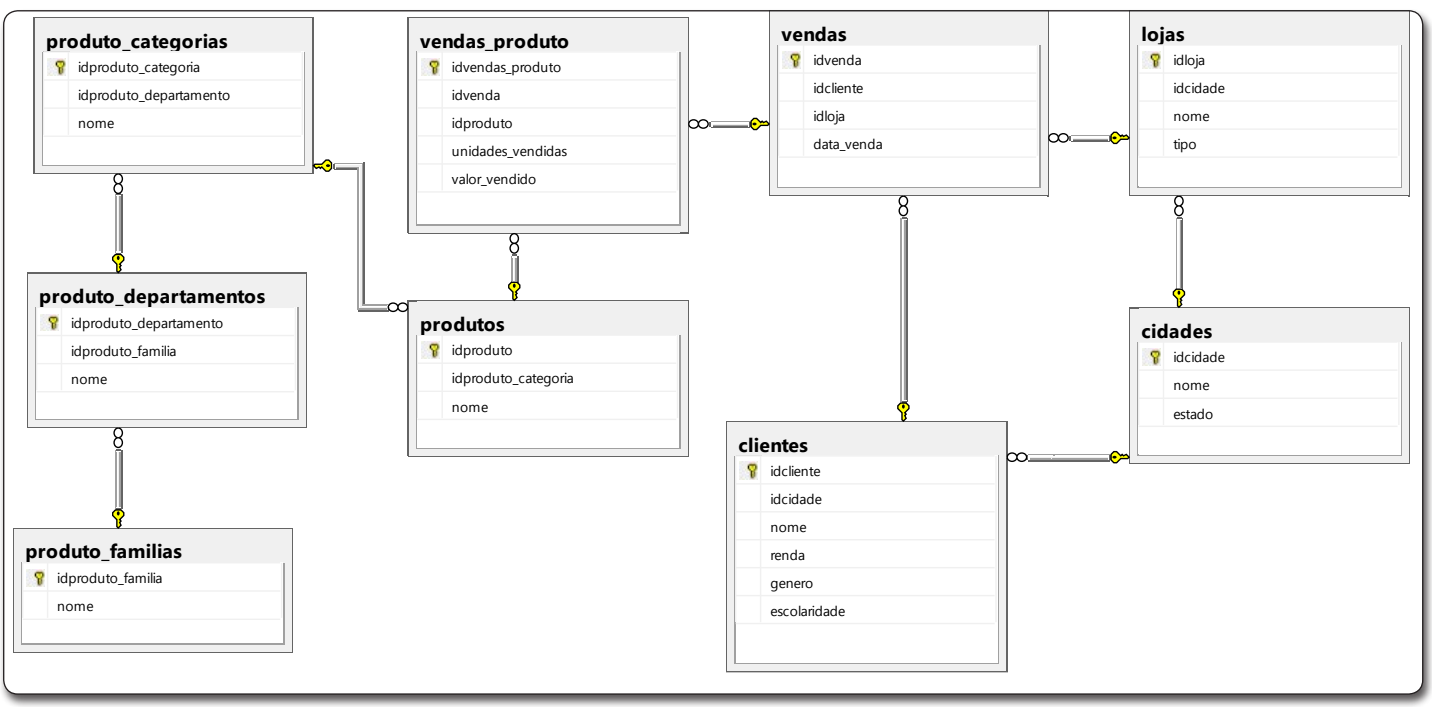


Figura 3. Diagrama ER da base de dados transaciona

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

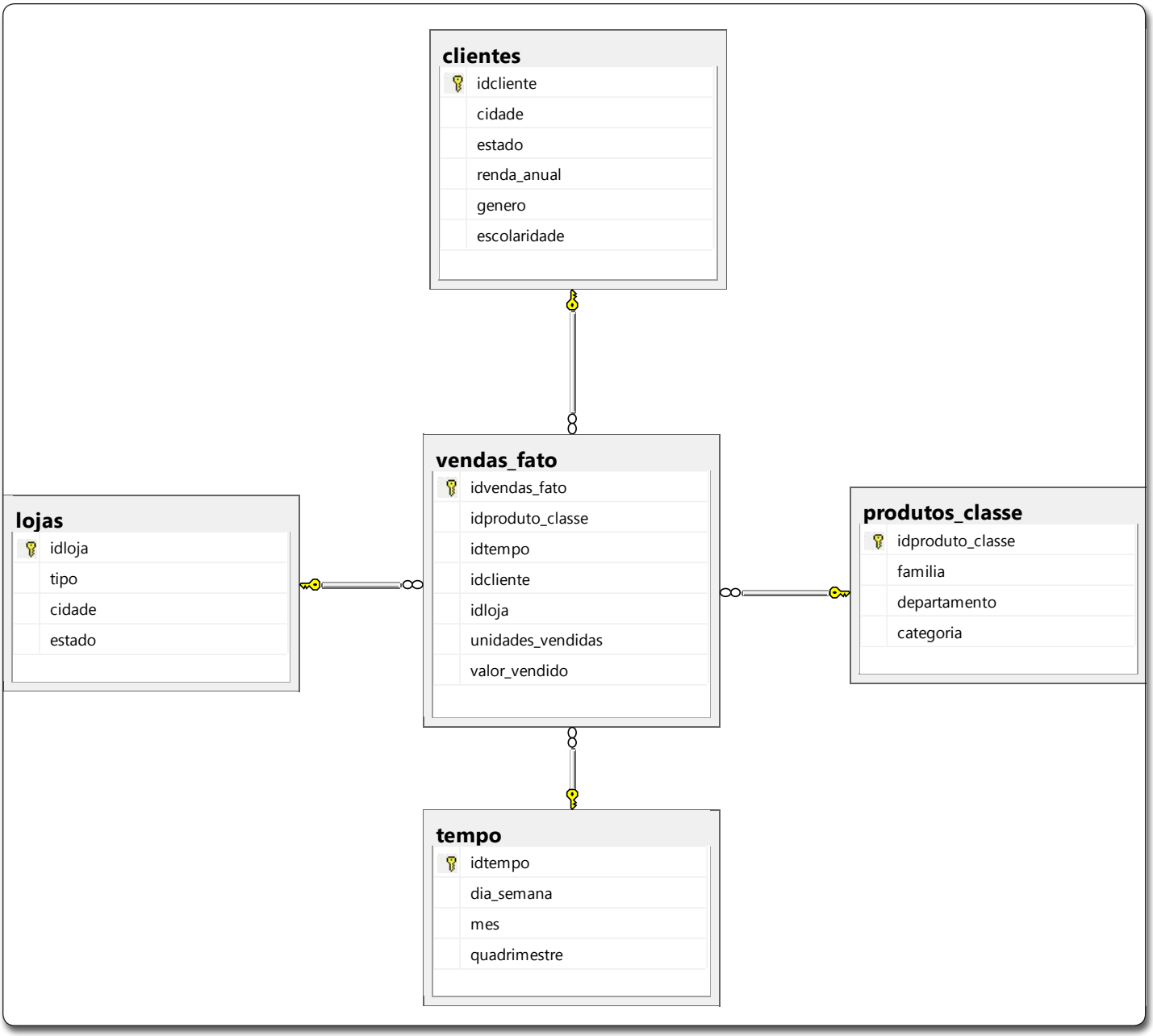


Figura 4. Diagrama ER do data warehouse

Por outro lado, há um pouco de desperdício de espaço pelo fato das mesmas descrições serem repetidas em todos os registros, como na tabela *produtos_classe*. A **Figura 4** é um exemplo de esquema de data warehouse que utiliza o modelo estrela.

Por outro lado, no modelo floco de neve as tabelas que representam as dimensões podem não estar ligadas diretamente à tabela fato. Por exemplo, na **Figura 4** poderia existir a tabela cidades relacionada com a tabela de clientes e de lojas ao invés deste campo ser armazenado em formato texto diretamente nessas tabelas. Este modelo é mais complexo e, em geral, consome mais recursos computacionais para realizar as consultas, já que mais *joins* em SQL são necessários.

Criando o data warehouse

Esta seção tem o objetivo de mostrar os comandos em SQL para a construção do data warehouse apresentado na **Figura 4**, bem como explanar os campos existentes em cada uma das dimensões. A **Listagem 1** mostra o script para a criação da dimensão *clientes*, e conforme abordado na seção anterior, os tipos de dados dos campos *cidade* e *estado* são caracteres, seguindo as definições do modelo estrela da modelagem multidimensional. A *renda_anual* também é representada por um campo alfanumérico e é composta por oito faixas de valores: R$ 0.0 – R$ 30.000, R$ 30.000 – R$ 50.000, R$ 50.000 – R$ 70.000, R$ 70.000 – R$ 90.000, R$ 90.000 – R$ 110.000, R$ 110.000 – R$ 130.000, R$ 130.000 – R$ 150.000 e R$ 150.000 +.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Na base de dados transacional é utilizado um campo numérico para indicar o valor exato da renda, porém, no DW se faz necessária a conversão desses valores numéricos em faixas para facilitar as análises e agrupamentos. Similarmente, o campo escolaridade é composto pelos seguintes valores: Ensino médio incompleto, Ensino médio completo, Ensino superior incompleto, Ensino superior completo e Pós-graduação completo. Por fim, o campo gênero é composto somente pelos valores M para masculino e F para feminino.

Listagem 1. SQL para criação da dimensão clientes

```
1 create table dwclientes (
2   idcliente int primary key not null,
3   cidade varchar(15),
4   estado varchar(10),
5   renda_anual varchar(25),
6   genero char(1),
7   escolaridade varchar(26))
```

Nota

Em geral, em aplicações comerciais é criada uma nova base de dados para armazenar o data warehouse. Neste exemplo optou-se por utilizar a mesma base por questões didáticas, e para diferenciar as tabelas está sendo utilizado o prefixo dw para indicar as tabelas que fazem parte de sua estrutura.

A **Listagem 2** apresenta o script para a criação da dimensão *lojas* e os campos *cidade* e *estado* seguem a mesma lógica apresentada para a dimensão *clientes*. A única diferença é o campo *tipo*, que armazena os seguintes valores: *Mercearia pequena*, *Mercearia grande*, *Supermercado*, *Supermercado de luxo*, *Supermercado gourmet* e *Sede da empresa*. Similarmente aos campos *cidade* e *estado*, o campo *tipo* também é do tipo textual.

A **Listagem 3**, por sua vez, apresenta o comando SQL para a criação da dimensão relacionada aos produtos, sendo composta somente pela família, departamento e categoria dos produtos, todos textuais. Esse fato foi explanado nas seções anteriores e segue as diretrizes do modelo estrela da modelagem multidimensional.

Listagem 2. SQL para criação da dimensão lojas

```
1 create table dwlojas (
2   idloja int primary key not null,
3   tipo varchar(20),
4   cidade varchar(15),
5   estado varchar(10))
```

Listagem 3. SQL para criação da dimensão classes de produtos

```
1 create table dwprodutos_classe (
2   idproduto_classe int primary key not null,
3   familia varchar(15),
4   departamento varchar(30),
5   categoria varchar(30))
```

A **Listagem 4** mostra o script para criação da dimensão *tempo*, que é composta pelo dia da semana (domingo até sábado), pelo mês (janeiro até dezembro) e pelo quadrimestre do ano (Q1 até Q4). Todos esses campos são alfanuméricos, com exceção da chave primária *idtempo* que é do tipo inteiro e possui a propriedade para auto incremento (*identity*). Pode-se notar que as outras dimensões não possuem auto incremento, e isso se deve ao fato de que nas outras tabelas o valor da chave primária será importado das próprias tabelas da base transacional. O auto incremento é utilizado na dimensão tempo porque essa tabela não existe originalmente na base de dados.

Por fim, a **Listagem 5** apresenta o comando SQL para a criação da tabela fato, a qual possui as medidas *unidades_vendidas* e *valor_vendido*, bem como as chaves estrangeiras que fazem o relacionamento com as dimensões criadas anteriormente. É importante frisar que neste modelo as medidas já apresentam os valores totalizados para cada venda. Por exemplo, caso a venda número *14* contenha cinco produtos, na tabela *dwvendas_fato* serão armazenadas as respectivas totalizações para a quantidade vendida e o valor total referente a esses cinco produtos. Desta forma, elimina-se a necessidade de utilizar a tabela *vendas_produto* da **Figura 3**.

Nota

Uma venda pode possuir vários produtos, porém, na tabela dwvendas_fato existe somente um campo idproduto, o qual armazenará somente o primeiro produto da tabela vendas_produto. Caso o projeto fosse de um data warehouse real, seria necessário um melhor tratamento desta questão.

Listagem 4. SQL para criação da dimensão tempo

```
1 create table dwtempo (
2   idtempo int primary key identity not null,
3   dia_semana varchar(15),
4   mes varchar(15),
5   quadrimestre char(2))
```

Listagem 5. SQL para criação da tabela fato

```
1 create table dwvendas_fato (
2   idvendas_fato int primary key not null,
3   idproduto_classe int foreign key references dwprodutos_classe
    (idproduto_classe),
4   idtempo int foreign key references dwtempo (idtempo),
5   idcliente int foreign key references dwclientes (idcliente),
6   idloja int foreign key references dwlojas (idloja),
7   unidades_vendidas int,
8   valor_vendido float )
```

Criando a interface gráfica e configurando o acesso aos dados no Delphi

As seções anteriores apresentaram a estrutura do banco de dados transacional e do data warehouse, bem como as principais diferenças entre ambos. Neste contexto, o objetivo das próximas seções é mostrar como importar os dados da base transacional e popular as dimensões e a tabela fato do DW. Esse procedimento será implementado via código e, para isso, a próxima etapa é a criação de uma aplicação do tipo VCL no Delphi (File>New>VCL Forms

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Application). A **Figura 5** apresenta a janela principal (*frmPrincipal*), a qual contém cinco componentes TButton (*btnClientes, btnLojas, btnProdutos, btnTempo e btnFato*), um componente TFDConnection (*conexao*) e dois componentes TFDQuery (*qryConsulta* e *qryInsercao*) da biblioteca FireDAC. Cada botão realizará a importação dos dados das tabelas correspondentes do modelo transacional para as tabelas de dimensões e para a tabela fato, enquanto que o componente de conexão é o responsável por realizar a ligação com a base de dados do SQL Server. Por outro lado, os componentes TFDQuery são os responsáveis pelas consultas e inserções que serão realiza1das posteriormente. Esses dois componentes devem ter a sua propriedade *Connection* setada para o componente *conexao*.

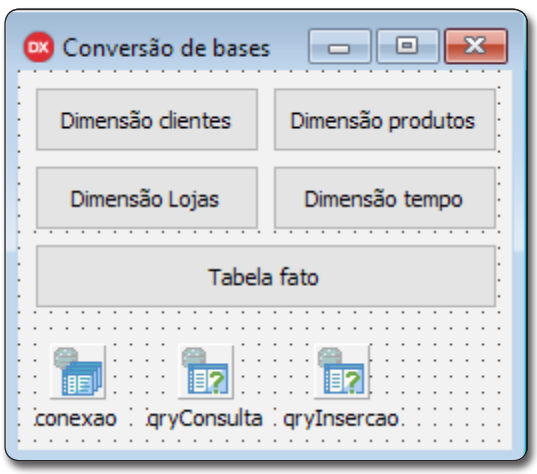


Figura 5. Componentes da janela

Duplo clique no componente de conexão abre o editor de conexão do FireDAC, o qual deve ter a sua propriedade *DriverID* configurada para *MSSQL*. Os outros parâmetros que devem ser preenchidos são o *Database* com o nome da base de dados (*Vendas*), o *Server* com o valor *localhost* e a propriedade *OSAuthent* deve estar com o valor *Yes*. O SQL Server possui integração com a segurança do próprio Windows, e esse parâmetro informa que usuários autenticados no sistema operacional terão acesso à base de dados. Caso a instalação do SQL Server tenha um usuário e senha específicos, os campos *User_Name* e *Password* devem ser preenchidos no editor de conexão do FireDAC. Adicionalmente, a propriedade *Login Prompt* do TFDConnection deve ser definida como *False* para que a janela de login e senha não seja aberta a cada tentativa de conexão. A **Figura 6** apresenta a janela de configuração.

Importando os clientes

O objetivo geral da importação dos dados é basicamente: (i) ler os dados da base transacional, (ii) realizar possíveis pré-processamentos e/ou ajustes nos dados e (iii) gravá-los no data warehouse. Para isso, a **Listagem 6** apresenta o código fonte do procedimento que será executado ao clicar no botão para importação dos dados dos clientes. Pode-se notar na linha 6 que é adicionado na propriedade *SQL* da *qryInsercao* o comando *insert* com parâmetros para inserir os registros na dimensão de clientes (*dwclientes*).

Por outro lado, na linha 7 é executada uma consulta que une a tabela de clientes com a tabela de cidades do modelo transacional, com o intuito de retornar os dados de acordo com as necessidades do data warehouse. Como existem 10.282 clientes nesta tabela, na linha 8 é definida uma estrutura de repetição para percorrer cada um dos registros existentes, executando assim os comandos para passagem dos valores retornados pela consulta por parâmetros. Entre as linhas 13 e 30 é executada uma operação de pré-processamento chamada de codificação, que consiste em mapear os valores numéricos existentes no campo *renda* do modelo transacional em intervalos. Esse processo é necessário devido ao fato de que existe uma grande variabilidade de rendas com valores únicos. Por exemplo, caso seja executado o comando SQL select distinct(renda) from clientes, serão obtidos 9.936 valores de rendas distintas, o que inviabilizaria a análise e agrupamentos dos clientes por este campo.

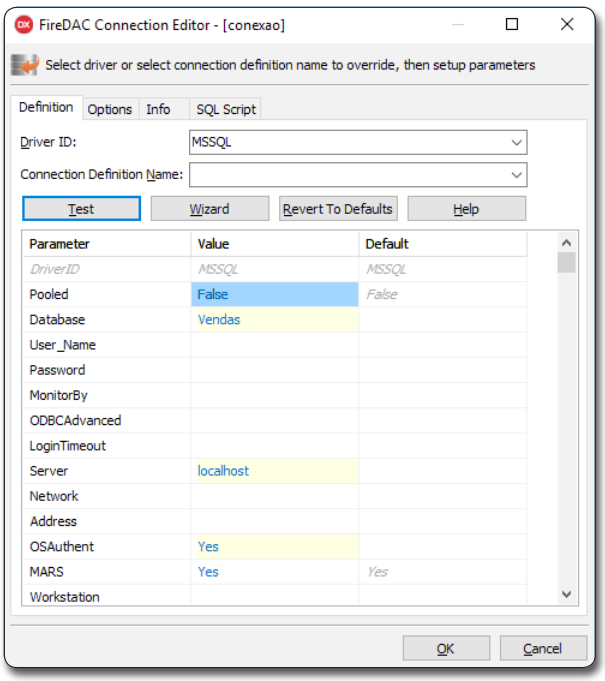


Figura 6. Configuração do acesso à base de dados

Nota

O mapeamento em intervalos realizado na dimensão de clientes é também conhecido como discretização. No exemplo proposto, as faixas de valores já estavam definidas, porém, métodos estatísticos de distribuição de frequência podem ser utilizados quando não se tem certeza sobre o melhor conjunto de intervalos.

Entre as linhas 32 e 38 da **Listagem 6** é também executada a codificação dos dados numéricos em alfanuméricos por meio do comando *case*. Na base de dados transacional, cada categoria de escolaridade é representada por um número inteiro, enquanto que neste data warehouse estão sendo armazenadas as categorias em formato textual. Por fim, na linha 40 a instrução insert é executada e na linha 41 o cursor da consulta aponta para o próximo cliente a ser inserido na dimensão cliente.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 6. Importação dos clientes

```
01 procedure TfrmPrincipal.btnClientesClick(Sender: TObject);
02 var
03  ARendaAnual: real;
04  ARotuloRendaAnual, ARotuloEscolaridade: string;
05 begin
06  qryInsercao.SQL.Add('insert into dwclientes (idcliente, cidade, estado,
     renda_anual, genero, escolaridade) values (:idcliente, :cidade, :estado,
     :renda_anual, :genero, :escolaridade)');
07  qryConsulta.Open('select idcliente, cd.nome as cidade, estado, renda,
     genero, escolaridade  from clientes cl left outer join cidades cd on cl.idcidade
     = cd.idcidade');
08  while not qryConsulta.Eof do
09  begin
10   qryInsercao.ParamByName('idcliente').AsInteger := qryConsulta.
      FieldByName('idcliente').AsInteger;
11   qryInsercao.ParamByName('cidade').AsString := qryConsulta.
      FieldByName('cidade').AsString;
12   qryInsercao.ParamByName('estado').AsString := qryConsulta.
      FieldByName('estado').AsString;
13   ARendaAnual := qryConsulta.FieldByName('renda').AsFloat;
14   if (ARendaAnual >= 0) and (ARendaAnual <= 30000) then
15    ARotuloRendaAnual := 'R$ 0.0 - R$ 30.000'
16   else if (ARendaAnual > 30000) and (ARendaAnual <= 50000) then
17    ARotuloRendaAnual := 'R$ 30.000 – R$ 50.000'
18   else if (ARendaAnual > 50000) and (ARendaAnual <= 70000) then
19    ARotuloRendaAnual := 'R$ 50.000 – R$ 70.000'
20   else if (ARendaAnual > 70000) and (ARendaAnual <= 90000) then
21    ARotuloRendaAnual := 'R$ 70.000 – R$ 90.000'
22   else if (ARendaAnual > 90000) and (ARendaAnual <= 110000) then
23    ARotuloRendaAnual := 'R$ 90.000 – R$ 110.000'
24   else if (ARendaAnual > 110000) and (ARendaAnual <= 130000) then
25    ARotuloRendaAnual := 'R$ 110.000 – R$ 130.000'
26   else if (ARendaAnual > 130000) and (ARendaAnual <= 150000) then
27    ARotuloRendaAnual := 'R$ 130.000 – R$ 150.000'
28   else if (ARendaAnual > 150000) then
29    ARotuloRendaAnual := 'R$ 150.000 +';
30   qryInsercao.ParamByName('renda_anual').AsString := ARotuloRendaAnual;
31   qryInsercao.ParamByName('genero').AsString := qryConsulta.
      FieldByName('genero').AsString;
32   case qryConsulta.FieldByName('escolaridade').AsInteger of
33    1: ARotuloEscolaridade := 'Ensino médio incompleto';
34    2: ARotuloEscolaridade := 'Ensino médio completo';
35    3: ARotuloEscolaridade := 'Ensino superior incompleto';
36    4: ARotuloEscolaridade := 'Ensino superior completo';
37    5: ARotuloEscolaridade := 'Pós-graduação completo';
38   end;
39   qryInsercao.ParamByName('escolaridade').AsString := ARotuloEscolaridade;
40   qryInsercao.ExecSQL;
41   qryConsulta.Next;
42  end;
43 end;
```

Importando as lojas

Semelhante à importação dos clientes, o procedimento para importar as lojas é bastante similar e é mostrado na **Listagem 7**. A única diferença é a codificação dos tipos de lojas em descrições completas entre as linhas 10 e 23. Pode-se observar que o campo *escolaridade* na importação dos clientes é similar, porém, é representado de forma numérica, enquanto que para as lojas essa representação apresenta-se no formato de siglas. Optou-se por essa representação para demonstrar que em bases de dados transacionais os campos nem sempre podem estar padronizados, fato que torna necessária uma análise mais detalhada dos campos das tabelas.

Listagem 7. Importação das lojas

```
01 procedure TfrmPrincipal.btnLojasClick(Sender: TObject);
02 var
03  ATipoLoja, ARotuloTipoLoja: string;
04 begin
05  qryInsercao.SQL.Add('insert into dwlojas (idloja, tipo, cidade, estado)
     values (:idloja, :tipo, :cidade, :estado)');
06  qryConsulta.Open('select idloja, cd.nome as cidade, cd.estado,
     tipo from lojas lj left outer join cidades cd on lj.idcidade = cd.idcidade');
07  while not qryConsulta.Eof do
08  begin
09   qryInsercao.ParamByName('idloja').AsInteger := qryConsulta.
      FieldByName('idloja').AsInteger;
10   ATipoLoja := qryConsulta.FieldByName('tipo').AsString;
11   if (ATipoLoja = 'MG') then
12    ARotuloTipoLoja := 'Mercearia grande'
13   else if (ATipoLoja = 'MP') then
14    ARotuloTipoLoja := 'Mercearia pequena'
15   else if (ATipoLoja = 'SE') then
16    ARotuloTipoLoja := 'Sede da empresa'
17   else if (ATipoLoja = 'SP') then
18    ARotuloTipoLoja := 'Supermercado'
19   else if (ATipoLoja = 'SL') then
20    ARotuloTipoLoja := 'Supermercado de luxo'
21   else if (ATipoLoja = 'SG') then
22    ARotuloTipoLoja := 'Supermercado gourmet';
23   qryInsercao.ParamByName('tipo').AsString := ARotuloTipoLoja;
24   qryInsercao.ParamByName('cidade').AsString := qryConsulta.
      FieldByName('cidade').AsString;
25   qryInsercao.ParamByName('estado').AsString := qryConsulta.
      FieldByName('estado').AsString;
26   qryInsercao.ExecSQL;
27   qryConsulta.Next;
28  end;
29 end;
```

Importando os produtos

A **Listagem 8** apresenta o código fonte para realizar a importação dos produtos. Na linha 3 foi declarada uma variável para armazenar a instrução SQL da consulta na base transacional. Pode-se observar na linha 5 que a consulta é longa, pois a seleção dos dados para o data warehouse é feita com base em todos os produtos existentes na base de dados (10.281 registros), e deve-se relacionar o produto com a categoria, a categoria com o departamento e o departamento com a família de produtos. Conforme abordado nas seções anteriores, o objetivo desta dimensão é apresentar os dados somente da categorização que existe entre os produtos. Isso se deve ao fato de que dependendo da base de dados, pode existir uma grande variedade de produtos, o que pode tornar as análises e agrupamentos mais difíceis para o analista.

Similar às importações anteriores, na linha 8 é definida uma estrutura de repetição para percorrer os 10.281 produtos existentes na tabela da base transacional, ou seja, a dimensão *dwprodutos_classe* conterá esse mesmo número de registros, porém, o nome do produto foi suprimido. Por esse motivo, na linha 10 a chave primária da dimensão recebe o mesmo valor da chave primária da tabela de produtos. Por meio dessa importação, posteriormente o analista poderá agrupar os dados por essas categorias, por exemplo: existem três famílias de produtos (select distinct(familia) from dwprodutos_classe), 19 departamentos (select distinct(departamento) from dwprodutos_classe) e 45 categorias (select distinct(categoria) from dwprodutos_classe).

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 8. Importação dos produtos

```
01 procedure TfrmPrincipal.btnProdutosClick(Sender: TObject);
02 var
03  ASQL: string;
04 begin
05  ASQL := 'select idproduto, pf.nome as familia, pd.nome as departamento,
    pc.nome as  categoria ' + ' from produtos pr inner join produto_categorias pc
    on pr.idproduto_categoria = pc.idproduto_categoria ' + ' inner join produto_
    departamentos pd on pc.idproduto_departamento = pd.idproduto_
    departamento inner join produto_familias pf on pd.idproduto_familia
    = pf.idproduto_familia';
06 qryInsercao.SQL.Add('insert into dwprodutos_classe (idproduto_classe, familia,
    departamento, categoria) values (:idproduto_classe, :familia, :departamento,
    :categoria)');
07 qryConsulta.Open(ASQL);
08 while not qryConsulta.Eof do
09 begin
10  qryInsercao.ParamByName('idproduto_classe').AsInteger := qryConsulta.
    FieldByName('idproduto').AsInteger;
11  qryInsercao.ParamByName('familia').AsString := qryConsulta.
    FieldByName('familia').AsString;
12  qryInsercao.ParamByName('departamento').AsString := qryConsulta.
    FieldByName('departamento').AsString;
13  qryInsercao.ParamByName('categoria').AsString := qryConsulta.
    FieldByName('categoria').AsString;
14  qryInsercao.ExecSQL;
15  qryConsulta.Next;
16 end;
17 end;
```

Importando os dados de tempo

Diferentemente das outras importações realizadas até o momento, para construir a dimensão *tempo* é necessário processar somente o campo *data_venda* da tabela de vendas do modelo transacional. Isso ocorre porque a dimensão tempo é formada somente pelo dia da semana, pelo mês e pelo quadrimestre, os quais são extraídos da data da venda. A **Listagem 9** apresenta o código necessário para realizar essa operação, no qual é possível observar na linha 4 que é utilizada a função *datepart* para extrair o dia da semana e o quadrimestre, e a função *month* para extrair o mês. Essas funções retornam um número inteiro que corresponde a cada período, por exemplo: considerando o campo dia da semana, o valor 1 representa o domingo e o 7 representa o sábado. Essa mesma lógica é também utilizada para o mês e o quadrimestre. Na instrução SQL da linha 4 é também possível observar que é utilizado o comando *distinct*, que tem a função de buscar da tabela de vendas somente os registros únicos que pertençam ao mesmo tempo a um dia da semana, a um mês e a um quadrimestre. Esse processo reduz a quantidade de registros únicos nessa dimensão para 84, conforme pode ser observado caso a consulta da linha 4 seja executada no editor de consultas do SQL Server.

Nota

A importação realizada na **Listagem 9** está gravando na dimensão tempo os valores numéricos relativos aos dias da semana, meses e quadrimestres. Um pré-processamento adicional é necessário para que esses valores sejam convertidos para dados textuais, por exemplo: domingo no lugar de 1. Esse processo não é abordado no código fonte por já ter sido mostrado anteriormente e por questões de espaço. Adicionalmente, o SQL Server apresenta variações dessas funções que retornam os valores em formato textual.

Listagem 9. Importação das datas

```
01 procedure TfrmPrincipal.btnTempoClick(Sender: TObject);
02 begin
03  qryInsercao.SQL.Add('insert into dwtempo (dia_semana, mes, quadrimestre)
    values (:dia_semana, :mes, :quadrimestre)');
04  qryConsulta.Open('select distinct datepart(dw, data_venda) as dia_semana,
    month(data_venda) as mes, datepart(qq, data_venda) as quadrimestre from
    vendas');
05  while not qryConsulta.Eof do
06  begin
07   qryInsercao.ParamByName('dia_semana').AsInteger := qryConsulta.
     FieldByName('dia_semana').AsInteger;
08   qryInsercao.ParamByName('mes').AsInteger := qryConsulta.
     FieldByName('mes').AsInteger;
09   qryInsercao.ParamByName('quadrimestre').AsInteger := qryConsulta.
     FieldByName('quadrimestre').AsInteger;
10  qryInsercao.ExecSQL;
11  qryConsulta.Next;
12 end;
13 end;
```

Importando as vendas (tabela fato)

A importação dos dados das vendas é um pouco mais complexa do que as dimensões, pois é preciso associar as chaves estrangeiras e realizar as totalizações na tabela *vendas_produto* do modelo transacional. Para isso, foi definida uma variável do tipo *record* e mais três funções auxiliares na seção, conforme mostrado na **Listagem 10**.

Listagem 10. Definição do record e das funções

```
01 type
02 TotalVenda = Record
03   UnidadesVendidas: Integer;
04   ValorTotal: Real;
05 End;

06 type
07  TfrmPrincipal = class(TForm)
  ...

08 private
09    function GetIdTempo(ADiaSemana, AMes, AQuadrimestre: Integer): Integer;
10   function GetIdProdutoClasse(AIdVenda: Integer): Integer;
11   function GetTotalVenda(AIdVenda: Integer): TotalVenda;
12 public
13 end;
```

Para gerar a implementação das funções, deve-se pressionar ao mesmo tempo as teclas CRTL+SHIFT+C, para que elas sejam criadas na seção *implementation* do arquivo. A **Listagem 11** apresenta o código das funções para buscar a classe do produto e os dados de tempo. A primeira função (*GetIdProdutoClasse*) recebe como parâmetro o identificador da venda (*idvenda*) e retorna a classe do produto relacionado a essa venda. Na linha 3 é criada uma query em tempo de execução, que executa o comando SQL definido na linha 6, retornando na linha 9 a classe do produto correspondente. Conforme explanado anteriormente, no projeto deste data warehouse é possível ligar uma venda somente a um produto, e por esse motivo na linha 6 é utilizado o comando *top 1* para retornar o primeiro produto retornado pela consulta.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 11. Funções para retornar valores

```
01 function TfrmPrincipal.GetIdProdutoClasse(AIdVenda: Integer): Integer;
02 begin
03  with TFDQuery.Create(nil) do
04  try
05   Connection := conexao;
06   SQL.Add('select idproduto_classe from dwprodutos_classe where
      idproduto_classe = (select top 1 idproduto from vendas_produto where
      idvenda = :idvenda)');
07   ParamByName('idvenda').AsInteger := AIdVenda;
08   Open();
09   Result := FieldByName('idproduto_classe').AsInteger;
10  finally
11   Free;
12  end;
13 end;

14 function TfrmPrincipal.GetIdTempo(ADiaSemana, AMes, AQuadrimestre:
    Integer): Integer;
15 begin
16  with TFDQuery.Create(nil) do
17  try
18   Connection := conexao;
19   SQL.Add('select idtempo from dwtempo where dia_semana = :dia_semana
      and mes = :mes and quadrimestre = :quadrimestre');
20   ParamByName('dia_semana').AsInteger := ADiaSemana;
21   ParamByName('mes').AsInteger := AMes;
22   ParamByName('quadrimestre').AsInteger := AQuadrimestre;
23   Open();
24   Result := FieldByName('idtempo').AsInteger;
25  finally
26   Free;
27  end;
28 end;
```

Similarmente, a função *GetIdTempo* recebe como parâmetro o dia da semana, o mês e o quadrimestre, retornando na linha 24 o campo *idtempo* da dimensão *tempo*. Isso é feito por meio da consulta definida na linha 19, que filtra os registros com base nos parâmetros recebidos. A **Listagem 12** apresenta o código que realiza a totalização da venda, pois como explanado anteriormente, a tabela fato armazena somente os totais das unidades vendidas e do valor vendido que são campos da tabela *vendas_produto* da **Figura 3**. Para isso, a função recebe como parâmetro o *idvenda* e retorna uma variável do tipo record *TotalVenda* que foi declarado na **Listagem 10**. O record possui dois campos: um para representar as unidades e outro para representar o valor total, e seu objetivo é facilitar o retorno da função. O comando SQL da linha 8 faz a soma das unidades vendidas, bem como realiza a multiplicação do valor vendido pelas unidades para obter o valor total. Nas linhas 12 e 13 o record recebe o resultado da consulta e na linha 14 é feito o retorno da função.

Por fim, a **Listagem 13** realiza o processo de importação das vendas. Na linha 7 é definido o comando SQL que faz a busca na tabela de vendas da base transacional, enquanto que na linha 8 é definido o comando para inserção dos dados. O processo é bastante similar às outras importações realizadas, com exceção das linhas 13 a 17, que buscam a classe do produto e associam ao parâmetro de inserção. Uma estrutura condicional foi feita para verificar se existe alguma venda sem produtos cadastrados, e,

Listagem 12. Função para totalizar as vendas

```
01 function TfrmPrincipal.GetTotalVenda(AIdVenda: Integer): TotalVenda;
02 var
03  ATotal: TotalVenda;
04 begin
05  with TFDQuery.Create(nil) do
06  try
07   Connection := conexao;
08   SQL.Add('select sum(unidades_vendidas) as unidades_vendidas,
      sum(valor_vendido * unidades_vendidas) as valor_vendido from vendas_
      produto where idvenda = :idvenda');
09   ParamByName('idvenda').AsInteger := AIdVenda;
10   Open();
12   ATotal.UnidadesVendidas := FieldByName('unidades_vendidas').AsInteger;
13   ATotal.ValorTotal := FieldByName('valor_vendido').AsFloat;
14   Result := ATotal;
15  finally
16   Free;
17  end;
18 end;
```

Listagem 13. Importando as vendas

```
01 procedure TfrmPrincipal.btnFatoClick(Sender: TObject);
02 var
03  ASQL: string;
04  ATotalVenda: TotalVenda;
05  AIdProdutoClasse: Integer;
06 begin
07  ASQL := 'select idvenda, idcliente, idloja, datepart(dw, data_venda)
     as dia_semana, month(data_venda) as mes, datepart(qq, data_venda)
     as quadrimestre from vendas';
08  qryInsercao.SQL.Add('insert into dwvendas_fato (idvendas_fato,
     idproduto_classe, idtempo, idcliente, idloja, unidades_vendidas,
     valor_vendido) values (:idvendas_fato, :idproduto_classe, :idtempo,
     :idcliente, :idloja, :unidades_vendidas, :valor_vendido)');
09  qryConsulta.Open(ASQL);
10  while not qryConsulta.Eof do
11  begin
12   qryInsercao.ParamByName('idvendas_fato').AsInteger := qryConsulta.
     FieldByName('idvenda').AsInteger;
13   AIdProdutoClasse := GetIdProdutoClasse(qryConsulta.
     FieldByName('idvenda').AsInteger);
14   if (AIdProdutoClasse > 0) then
15    qryInsercao.ParamByName('idproduto_classe').
      AsInteger := AIdProdutoClasse
16   else
17    qryInsercao.ParamByName('idproduto_classe').Value := null;
18   qryInsercao.ParamByName('idtempo').AsInteger := GetIdTempo(qryConsulta.
     FieldByName('dia_semana').AsInteger, qryConsulta.FieldByName('mes').
     AsInteger, qryConsulta.FieldByName('quadrimestre').AsInteger);
19   qryInsercao.ParamByName('idcliente').AsInteger := qryConsulta.
     FieldByName('idcliente').AsInteger;
20   qryInsercao.ParamByName('idloja').AsInteger := qryConsulta.
     FieldByName('idloja').AsInteger;
21   ATotalVenda := GetTotalVenda(qryConsulta.FieldByName('idvenda')
     .AsInteger);
22   qryInsercao.ParamByName('unidades_vendidas').AsInteger := ATotalVenda.
     UnidadesVendidas;
23   qryInsercao.ParamByName('valor_vendido').AsFloat := ATotalVenda.ValorTotal;
24   qryInsercao.ExecSQL;
25   qryConsulta.Next;
26  end;
27 end;
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No fórum da DevMedia você irá encontrar uma equipe disponível e altamente qualificada com consultores e colaboradores prontos para te ajudar a qualquer hora e sobre qualquer assunto. Temos as salas de Java, .NET, Delphi, Banco de Dados, Engenharia de Software, PHP, Java Script, Web Design, Automação comercial, Ruby on Rails e muito mais!

ACESSE AGORA
www.devmedia.com.br/forum

DEVMEDIA

nesta situação, o parâmetro da classe do produto é setado para nulo. Neste exemplo estão sendo carregados para o data warehouse todas as vendas, independente de terem produtos associados. Na base de dados de exemplo, existem mais de 100 vendas nesta situação, e isso pode gerar problemas de interpretação ou até mesmo representar inconsistências presentes na base de dados transacional. Em uma aplicação comercial, esses dados teriam que ser avaliados pelo analista, e possivelmente ajustados ou excluídos. Entre as linhas 18 e 23 são chamadas as funções criadas nas **Listagens 11** e **12**, as quais retornam o cliente, a loja e o tempo.

A importação de dados de uma base transacional para um data warehouse pode ser uma tarefa trabalhosa, principalmente se a quantidade de dimensões e registros for grande. Como visto no exemplo desenvolvido, são necessários vários procedimentos para realizar essa operação. Somente devem ser carregados no DW dados consistentes e que possam representar algum tipo de informação útil, portanto, em bases comerciais deve-se tomar um maior cuidado antes de transferir os dados. Neste contexto, diversas outras formas de pré-processamento podem ser realizadas, bem como a aplicação de algoritmos de filtros para escolher quais são os campos mais relevantes, a identificação e a redução de hierarquias entre os atributos, a limpeza de dados inconsistentes, o tratamento de informações ausentes e a normalização. Alguns desses métodos ou todos eles podem e devem ser aplicados, dependendo da necessidade da base de dados.

Referências:

J. Granatyr, E. E. Scalabrin, F. A. Taffe, A. D. Darold, "Business Intelligence utilizando o Fast Cube", Clube Delphi, v. 166, 2016.

R. Angelantonio, "Processo de modelagem de Data Warehouse", SQL Magazine, v. 131, 2015.

A. G. D. Ferreira, "Como definir um Data Warehouse na prática usando Transact SQL", v. 130, 2015.

R. Goldschmidt, E. Passos, "Data Mining: um guia prático", Editora Campus, 2005.

F. N. R. Machado, "Tecnologia e Projeto de Data Warehouse", Editora Érica, 2004.

Autor

Jones Granatyr

Doutorando em Informática e Mestre em Ciência da Computação, ambos na área de Inteligência Artificial. Trabalha em projetos de pesquisa relacionados à área de Inteligência Artificial, tais como Sistemas Especialistas, Mineração de Dados, Mineração de Textos e Sistemas Multiagente.

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Introdução ao desenvolvimento web no Delphi

Conheça algumas formas de criar aplicações web no Delphi

Devido à grande evolução da Web nos últimos anos, temos observado um amplo movimento de aplicações que antes só era possível se desenvolver para desktop, devido à sua complexidade e formas de interação com usuários, sendo desenvolvidas para a internet.

À medida que os usuários intensificavam o uso da internet, a necessidade do acesso a dados e funcionalidades a partir de locais distintos, onde muitas vezes não era possível ou viável ter uma aplicação desktop instalada, também se tornava maior. Aliado a isso, requisitos e possibilidades como escalabilidade, manutenção centralizada e, mais recentemente, a evolução da computação nas nuvens fizeram com que as aplicações web ganhassem considerável importância no mercado de software. Nos dias atuais, inclusive, a criação de novos projetos apenas para ambiente desktop tem reduzido consideravelmente, devido à mudança no perfil dos usuários, sejam eles individuais ou empresas.

Uma das primeiras abordagens para alcançar esse objetivo era chamada de RIA (*Rich Internet Applications*), aplicações web que tentavam reproduzir os padrões de interface normalmente encontrados em ambiente desktop. Esse modelo se mostrava interessante principalmente por que requeria dos desenvolvedores poucas mudanças nas técnicas geralmente utilizadas em projetos desktop, bem como oferecia aos clientes finais uma experiência de uso muito próxima daquela com a qual estavam acostumados.

Independentemente do tipo de interface que será criado, desenvolver essas aplicações geralmente re-

Fique por Dentro

Nesse artigo será visto como podem ser desenvolvidas aplicações web utilizando o Delphi. O leitor verá como é possível desenvolver para a internet sem a necessidade de ter conhecimento de outra linguagem server-side, como PHP ou Java, e até mesmo linguagens client-side como HTML. Este assunto é útil para quem deseja distribuir suas aplicações de forma diferente, indo além dos sistemas desktops tradicionalmente objetivados por quem usa Delphi. Além disso, os recursos que serão abordados neste artigo são úteis também por permitirem a criação de projetos web de forma muito semelhante àqueles para desktop, reduzindo a necessidade de conhecer novas técnicas e linguagens de desenvolvimento.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

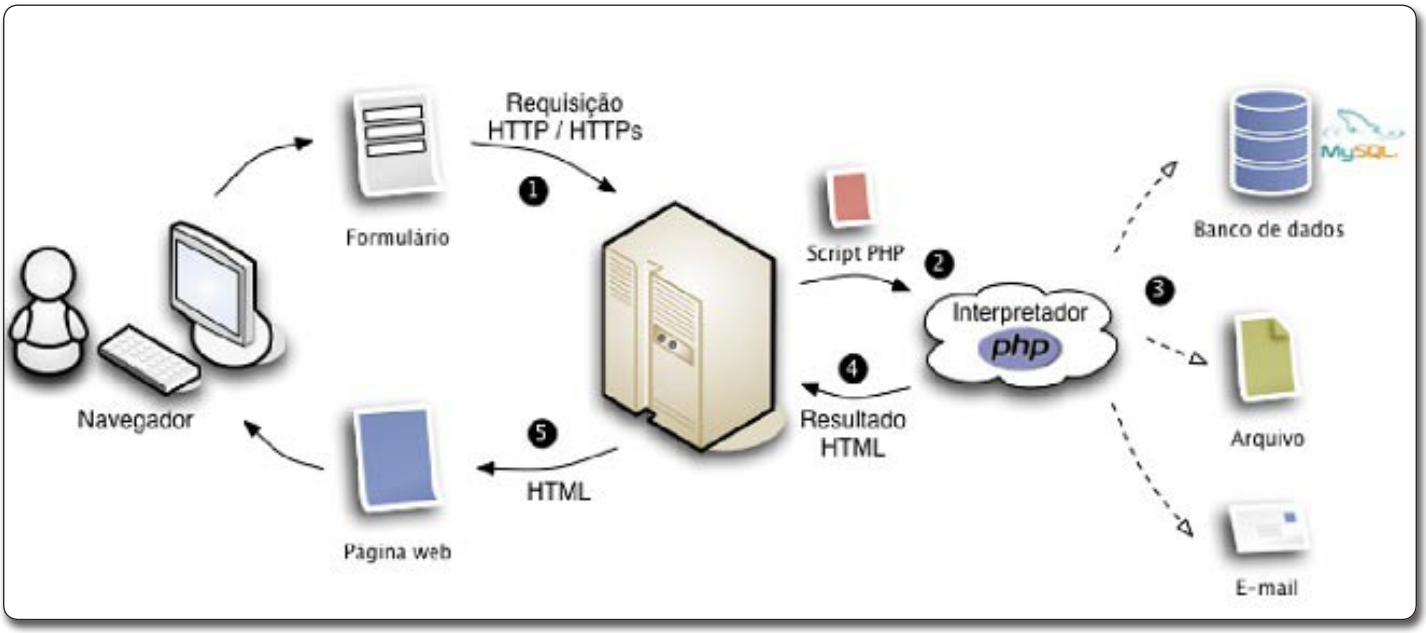


Figura 1. Representação do funcionamento de uma aplicação web

quer o conhecimento de uma linguagem *server-side*, além daquelas utilizadas no *client-side*. Isso, no entanto, pode se tornar uma dificuldade para desenvolvedores autônomos ou mesmo equipes que, por algum motivo, não tenham disponibilidade (ou interesse) para estudar tecnologias diferentes daquelas com as quais já têm experiência.

Para acompanhar essa evolução, o Delphi conta com recursos para permitir o desenvolvimento de aplicações web a partir do conhecimento e técnicas já aplicados nos projetos desktop. Alguns desses recursos fazem parcialmente e outros completamente a abstração de técnicas ou linguagens *server-side* e *client-side*, evitando a necessidade de utilizar outras linguagens de programação.

Neste artigo conheceremos, além de recursos disponíveis de forma nativa no Delphi, como o Intraweb e DataSnap, também frameworks como o UniGUI e Raudus para o desenvolvimento de soluções web.

Como funciona uma aplicação web

Uma aplicação web, diferente daquelas para desktop, têm seu processamento em um servidor na Web e não localmente. O cliente, ao acessar o URL (*Uniform Resource Locator*) desejado, faz uma requisição a um servidor, que a processa e envia como resposta um arquivo, geralmente HTML. O tipo desse retorno (HTML, XML, JSON, etc.) determina como o browser deve apresentar seu conteúdo.

Ao receber uma requisição do cliente, o servidor processa as informações, consultando-as e persistindo-as no banco de dados, arquivos, etc. Em seguida um resultado é retornado para o cliente e pode conter diversos tipos de arquivos, imagens, etc. A **Figura 1** ilustra esse processo.

Server-side

Server-side, também conhecido como *back-end*, é o termo utilizado para operações realizadas no lado do servidor em uma arquitetura do tipo *client-server*. Neste lado da arquitetura são realizadas operações como acesso ao banco de dados, autenticação e autorização. Esse tipo de processamento dá aos sites e aplicações web o dinamismo do qual necessitam, pois sem isso as páginas na internet seriam puramente estáticas, apenas compostas por linguagens do *front-end*.

Entre as linguagens mais utilizadas no *back-end* das aplicações estão Java, C# (com o ASP.NET), PHP, dentre outras. Estas possuem recursos para tratar as requisições HTTP, geralmente enviadas através de navegadores da internet, e retornar algum resultado que seja processável pelo cliente, como uma página HTML, um arquivo, um documento JSON ou XML.

Client-side

Client-side, comumente chamado também de *front-end*, é o termo utilizado para operações realizadas no lado do cliente, ou

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

seja, aquelas que geralmente são feitas no browser. A cargo das linguagens desse lado ficam a construção das interfaces gráficas utilizadas pelos usuários ao acessarem uma página na Web, bem como comportamentos típicos desse lado da arquitetura, como validação de formulários.

Intraweb

Intraweb é uma biblioteca de componentes para desenvolvimento de aplicações dinâmicas para web disponível para Delphi, Kylix e Visual Basic .NET. Estas, diferente das aplicações tradicionais para web, geram apenas um arquivo contendo todo o projeto desenvolvido, com exceção da base de dados e figuras. Ou seja, não temos arquivos HTML, CSS e JS para cada página.

Ao criar um projeto Intraweb deve-se somente utilizar componentes das paletas dessa biblioteca, devido ao fato de eles serem renderizados como HTML. Os demais componentes visuais não surtiriam o efeito esperado se utilizados nessas aplicações.

No Delphi existem quatro paletas de componentes específicas para trabalhar com esse tipo de projeto:

- **IntraWeb Standard**: Esta paleta se assemelha à Standard da VCL (*Visual Component Library*). Nela estão localizados os componentes mais simples do Intraweb, como labels, buttons, comboboxes, timers, grids, checkboxes, menus, além de outros específicos do ambiente web, como link, URL, applet, flash, layers, etc.;
- **IntraWeb Data**: Semelhante à antiga paleta Data Controls da VCL. Aqui estão localizados os componentes para controle da base de dados, como DBGrid, DBLabel, DBEdit, DBNavigator. Assim como os da VCL, eles são ligados com um Dataset;
- **IntraWeb Control**: Nesta paleta estão componentes para controle da página a ser exibida, como layouts (templates) e produtores de páginas;
- **IntraWeb Authentication**: Nesta paleta estão componentes para controle da política de segurança.

O Intraweb oferece suporte para utilização Stand Alone (aquelas que possuem um servidor interno) no Windows ou hospedagem nos serviços do ISS ou do Apache.

Criaremos agora uma aplicação com essa biblioteca para ver, na prática, seu funcionamento. Para isso, vá no menu File > New > Other e na aba Intraweb selecione Intraweb Application Wizard. Para aplicações Stand Alone selecione Stand Alone Application, e se desejar hospedar a aplicação em um servidor ISS ou Apache selecione a opção ISAPI Extension.

Em Project Name digite *WebApplicationIntraWeb* e em Project Directory defina o diretório do projeto.

Um novo projeto Intraweb é composto inicialmente por três units:

- *ServerController.pas*: Nesse formulário estão configurações como porta, nome da aplicação, *time-out*, temas e configuração de cache. Essa unit serve para controle global da aplicação e nela é declarada e implementada a classe *TIWServerController*, onde podem ser adicionadas propriedades de nível global, valores irão refletir em todos os usuários da aplicação. Sempre que for necessário recuperar dados dessa classe pode-se utilizar o método *IWServerController*, que retornará uma instância da *TIWServerController* ativa;
- *UserSessionUnit.pas*: Essa é uma unit de controle de seção e conta com a classe *TIWUserSession*, à qual podem ser adicionadas propriedades a serem utilizadas durante a seção. Para recuperar os dados da seção armazenados por esta classe utiliza-se método *UserSession*, que retorna uma instância com os dados da seção atual.;
- *Unit1.pas*: Essa unit é o formulário principal do projeto, onde serão implementados os componentes visuais de interação com o usuário. Caso queira adicionar novos formulários, utilize o menu File > New > Other e na aba Intraweb selecione a opção New Form;

Adicione agora ao formulário um TIWLabel, um TIWEdit e um TIWButton. Em seguida, altere o caption do TIWLabel para "Informe seu nome". Remova o conteúdo da propriedade Text do TIWEdit e altere seu nome para edtNome. No botão, altere o Caption para "Ola" e adicione o seguinte método, que apenas exibe uma mensagem contendo o texto do edit:

```
WebApplication.ShowMessage('Ola ' + edtNome.Text);
```

Salve e execute o projeto. Será apresenta uma tela como a da **Figura 2**, onde constam informações sobre a versão da biblioteca e a porta onde a aplicação está sendo executada. Nessa tela clique no ícone do navegador em que deseja executar a aplicação.

Utilizando o navegador da internet (Google Chrome, neste caso) abra a aplicação e pressione F12 (Ferramentas do Desenvolvedor). Ao analisar o código fonte da aplicação, como mostra a **Figura 3**, vemos que esse consiste basicamente de uma página HTML.

Pode-se observar que toda a estrutura criada no Delphi foi renderizada na forma HTML, CSS e JavaScript no browser. Isso ocorre porque o serviço de resposta implementado pelo Intraweb faz com que as respostas enviadas ao browser contenham arquivos que ele consiga interpretar.

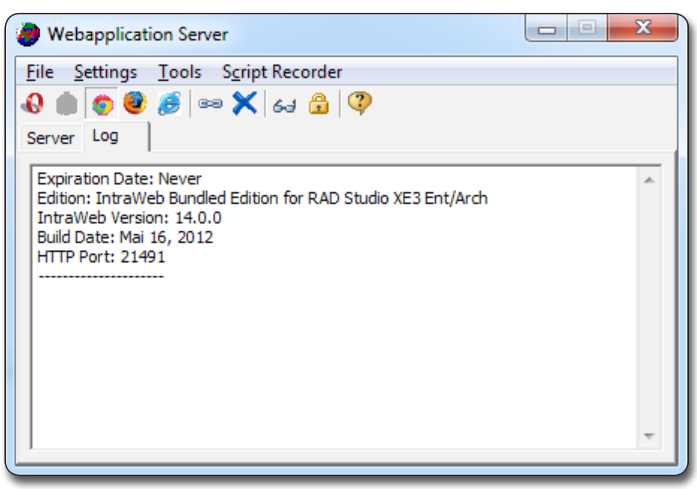


Figura 2. Servidor StandAlone

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

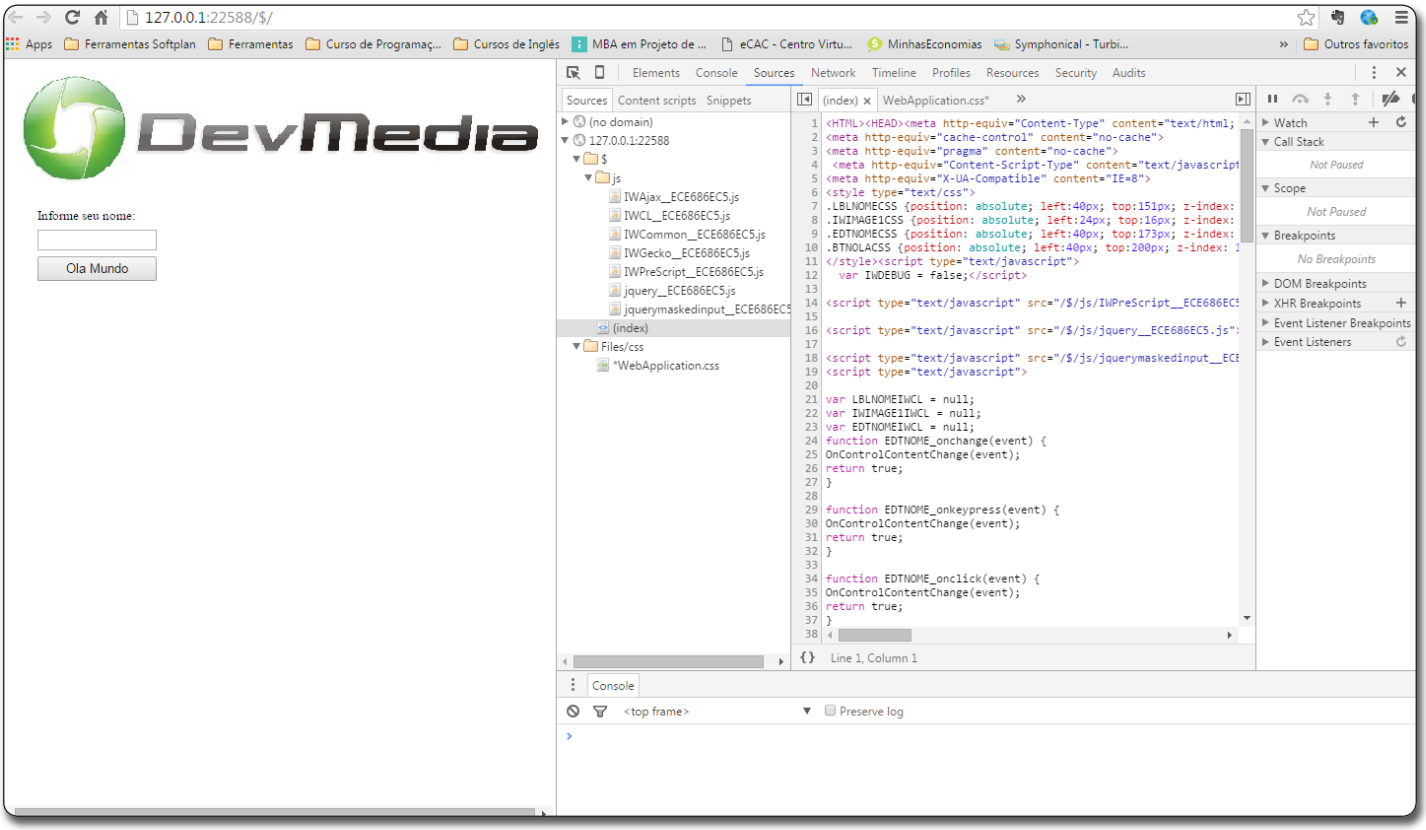


Figura 3. Retorno do servidor Stand Alone de Intraweb

Como vimos, a utilização da Intraweb abstraiu completamente a necessidade de ser ter algum conhecimento adicional de qualquer linguagem utilizada na web. No entanto, o conhecimento dessas tecnologias nos permite criar aplicações mais completas, com uso de JavaScript e CSS para melhorar o visual e comportamento das páginas.

Por padrão, o visual da aplicação fica bem semelhante a uma aplicação desktop, mas isso pode ser melhorado com o uso de CSS. Para isso, dentro da pasta de *output* da aplicação crie a seguinte estrutura de diretórios: *wwwroot\Files\css*. Na pasta *css* crie um arquivo chamado *WebApplication.css* e implemente as linhas de código que podem ser vistas na **Listagem 1**.

Listagem 1. WebApplication.css para Intraweb

```
01 input[type=button] {
02     cursor:pointer;
03     background:#35b128;
04     border:1px solid #33842a;
05    -webkit-border-radius: 10px;
06    border-radius: 10px;
07        color:#f3f3f3;
08        font-size:1.1em;
09 }
```

No formulário *ServerController*, na propriedade *StyleSheet.Filename*, coloque o valor *"\css\WebApplication.css"*. Isso vai fazer com que a aplicação considere o arquivo de CSS criado na pasta *css*. Na sequência salve e compile o projeto.

Como resultado temos um botão diferenciado na cor verde, com cantos arredondados e com fonte branca, como vemos na **Figura 4**.

Geralmente os programadores desktop não precisam preocupar-se com esse tipo de formatação visual, então esse trabalho pode ser atribuído a outro profissional com maior conhecimento nessa área, separando melhor as responsabilidades entre os componentes da aplicação (apresentação e lógica).

UniGUI

UniGUI (*Unified Graphical User Interface*) é um framework desenvolvido sobre o ExtJS (**BOX 1**) que permite o desenvolvimento

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

de aplicações web no Delphi. Seu conjunto de componentes permite criar aplicações web também sem a necessidade de conhecimento de linguagens para a web.

BOX 1. ExtJS

É um framework utilizado para a construção de aplicativos web interativos utilizando as linguagens Ajax, DHTML e a plataforma DOM. Este framework utiliza a arquitetura MVC como padrão de desenvolvimento, oferecendo diversos componentes de UI (User Interface) comuns às principais aplicações comerciais, como grids, formulários, botões, comboboxes, checkboxes.

Figura 4. Aplicando CSS no IntraWeb

O desenvolvimento de uma aplicação web a partir do UniGUI se assemelha ao uso da Intraweb e seus componentes são limitados às das seguintes paletas:

• **UniGUI Standard**: Se assemelha à paleta Standard da VCL. Nesta estão localizados os componentes mais simples, como labels, buttons, comboboxes, timers, grids, checkboxes, menus;

• **UniGUI Additional**: Semelhante à paleta Additional da VCL, nesta estão localizadas melhorias de componentes e componentes adicionais para implementação de frames, upload e download;

• **UniGUI Extra**: Nesta paleta encontram-se componentes de timer, calendário e gráficos;

• **UniGUI DataControls**: Semelhante à antiga paleta Data Controls da VCL, nesta localizados os componentes para controle da base de dados, como DBGrid, DBLabel, DBEdit, DBNavigator. Assim como os da VCL, eles são ligados com um Dataset.

O UniGUI oferece suporte para hospedagem diretamente como aplicação do Windows (Stand Alone) ou nos serviços do ISS ou do Apache, assim como a Intraweb.

Para melhor compreender esse framework, veremos agora como desenvolver uma aplicação utilizando-o. Assim como no exemplo anterior, acesse o menu File>New>Other e na aba UniGUI for Delphi selecione Application Wizard.

Para aplicações Stand Alone selecione StandAlone Server e se desejar hospedar a aplicação em um servidor ISS ou Apache selecione a opção ISAPI Module.

Em Project Name digite *WebApplicationUniGUI* e em Project Folder defina o diretório do projeto.

Um novo projeto UniGUI é composto inicialmente por três units:

• *ServerModule.pas*: Nesse formulário estão configurações como porta, nome da aplicação, *time-out*, temas, caminho do ExtJS, configuração de cache e DisplayMode. Essa unit serve para controle global da aplicação, e nela é declarada e implementada a classe TUniServerModule, onde podem ser adicionadas propriedades globais. Para recuperar dados dessa classe pode-se utilizar o método *UniServerModule* que retornará uma instância da *TUniServerModule* ativa;

• *MainModule.pas*: Nesse formulário é feito o controle de seção. Essa unit conta com a classe TUniMainModule, à qual podem ser adicionadas propriedades a serem utilizadas durante a seção. Para recuperar os dados armazenados por esta classe utiliza-se o método *UniMainModule*, que retorna uma instância com os dados da seção atual;

• *Main.pas*: É o formulário principal do projeto, onde serão adicionados os componentes visuais de interação com o usuário. Caso queira adicionar novos formulários, vá no menu File>New>Other e na aba UniGUI selecione Form.

Nota

A pasta ExtJS, que se encontra no diretório de instalação do UniGUI (\FMSoft\Framework\uniGUI\ext-...), deve ser distribuída juntamente a aplicação e deve ser determinado seu caminho na propriedade ExtRoot do ServerModule para seu correto funcionamento.

Seguindo com o exemplo, adicione ao formulário principal um TUniLabel, um TUniEdit e um TUniButton. Altere o Caption do TUniLabel para "Informe seu nome", limpe o Text do edit e altere seu nome para edtNome. Em seguida, altere o Caption para "Ola" e adicione o seguinte método, que exibirá uma mensagem contendo o texto do edit:

```
ShowMessage('Ola ' + edtNome.Text);
```

Salve e compile o projeto. A aplicação será executada na bandeja da barra de tarefas com um *tray icon* a lado do relógio. Clique com o botão direito sobre ele e selecione a opção Restore. Será aberta uma janela mostrando o tempo que a aplicação está em execução.

Na janela do server do projeto vá até Manage e clique em Server Monitor. Será aberto o *browser status* do servidor com o número de conexões ativas, número de requests, dados trafegados e recursos utilizados.

Para abrir a aplicação no navegador digite o endereço *http://localhost:<porta>/* onde <porta> é a porta de acesso ao serviço e pode ser definida na propriedade *Port* do *ServerModule*.

Assim como no desenvolvimento da aplicação utilizando o Intraweb, os passos são semelhantes ao desenvolvimento de uma aplicação desktop no Delphi. Nesse caso, inclusive o *ShowMessage* foi abstraído. No entanto, existem particularidades naturalmente são conhecidas com a experiência. Uma delas é o uso de *ShowModal*, que em aplicações web não para o processamento, como em aplicações desktop.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

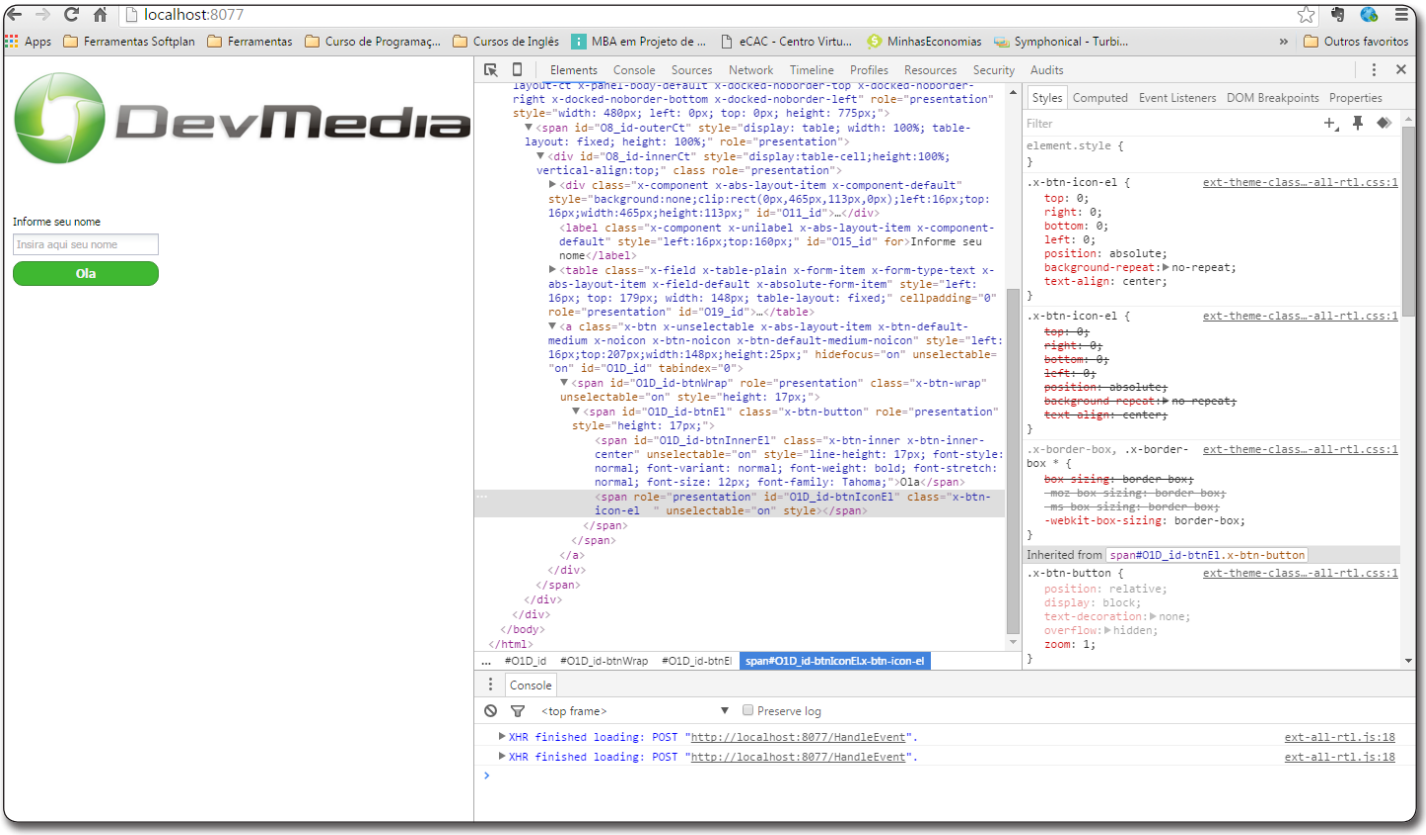


Figura 5. Retorno do servidor Stand Alone de UniGUI

O visual de uma aplicação desenvolvida com UniGUI pode ser inicialmente mais agradável que aquelas criadas com Intraweb, sem utilização de CSS, e se assemelham a uma aplicação desktop rodando dentro de um navegador. Por padrão, pode ser observado que a aplicação apresenta uma janela flutuante dentro no navegador, comportamento que pode ser alterado ao definir a propriedade *MainFormDisplayMode* para *mfPage* no *ServerModule*. De forma complementar, para alterar o skin da aplicação, modifique a propriedade *Theme* do *MainModule* para o estilo desejado.

Assim como na Intraweb, podemos utilizar recursos do CSS para melhorar o visual das páginas. Isso pode ser feito de duas formas: alterar os arquivos de estilos desejados da pasta ExtJS distribuída, ou adicionar os arquivos CSS desejados, esses arquivos podem ser adicionados na propriedade *CustomFiles* do *ServerModule*. Os arquivos devem ser distribuídos juntamente com a aplicação e, para isso, pode-se criar pastas para melhor organizar esses arquivos. No projeto, crie uma pasta chamada *css* dentro do diretório *files* na pasta de output da aplicação. Nela crie um arquivo com nome de *WebApplication.css* e adicione o código da **Listagem 2**. Altere a propriedade *CustomFiles* do *ServerModule* para *.\files\css\WebApplication.css* e em seguida, salve e compile o projeto.

O resultado visual será igual ao da aplicação desenvolvida com Intraweb, no entanto, um aspecto diferente no desenvolvimento com o UniGUI é o CSS, que foi dividido em duas partes: uma para uma classe *x-btn*, que é o botão, e outra para o *x-btn-default-medium* com *x-btn-inner*. Isso se deve à forma com que o UniGUI estrutura sua resposta em HTML, criando uma estrutura muito grande de elementos HTML para representar um componente, como pode ser observado na **Figura 5**. Isso pode dificultar a aplicação de CSS devido à quantidade de elementos em cadeia, e assim como no desenvolvimento com Intraweb, esse trabalho pode ficar a cargo de um especialista na área.

O UniGUI disponibiliza na maioria de seus componentes a propriedade *ClientEvents*, que é composta por outras *subpropriedades* que nos permitem estender funcionalidades além

Listagem 2. WebApplicatio.css para UniGUI

```
01 .x-btn-default-medium .x-btn-inner {
02       color:#f3f3f3;
03       font-size:1.1em;
04 }
05
06 .x-btn {
07       cursor:pointer;
08         background:#35b128;
09         border:1px solid #33842a;
10         -webkit-border-radius: 10px;
11         border-radius: 10px;
12 }
```


DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Conhecimento faz diferença!

Faça já sua assinatura digital ! | www.devmedia.com.br/es

Faça um *upgrade* em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional.
Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software.
Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**

DEVMEDIA

do Object Pascal e controlar ações que são disparadas no lado cliente. A implementação desses eventos enriquece a interação com o usuário, contudo deve ser escrita em JavaScript e para isso se faz necessário conhecimento dessa linguagem.

O UniGUI disponibiliza uma série de demonstrações da utilização de vários de seus recursos, que ficam contidos na pasta de instalação (*Framework**uniGUI**Demos*). Nesses demos há exemplos de como utilizar jQuery nas aplicações, acesso ao Google Maps, visualização de PDF, dentre diversos recursos que podem ser aplicados no desenvolvimento de aplicações web.

Raudus

Raudus é um framework para desenvolvimento de aplicações web no Delphi que, assim como o UniGUI, utiliza a biblioteca ExtJS. Ele disponibiliza um conjunto de componentes para Delphi e Lazarus, que permitem criar *Rich Internet Applications* com processo de desenvolvimento muito semelhante ao de aplicações desktop, também sem necessidade de utilização direta de HTML, CSS e JavaScript.

Nele, assim como na Intraweb e UniGUI, devem ser utilizados apenas os componentes visuais de sua paleta. Esse framework conta com um número bem limitado de componentes dispostos em uma única paleta. Também de forma semelhante às opções anteriores, o Raudus oferece suporte para hospedagem como aplicação Windows ou em servidor Apache.

Para desenvolver uma aplicação com Raudus acesse o menu File>New>Other e na aba Raudus escolha uma das opções disponíveis: RaApplication (para hospedagem em uma aplicação Windows) ou RaApplication for Apache (para hospedagem em um servidor Apache).

Um projeto Raudus, diferente da Intraweb e do UniGUI, é composto por apenas uma unit e a *main* onde será implementada toda a parte de view para o cliente. Configurações globais, como a porta de acesso, podem ser feitas na propriedade *Application.Config* utilizando a subpropriedade que desejar.

Adicione ao formulário um TRaLabel, um TRaEdit e um TRaButton. Em seguida, altere o Caption do TRaLabel para "Informe seu nome", remova o Text do edit e altere seu nome para edtNome. No botão, altere o Caption para "Ola".

O Raudus não implementa um ShowMessage e, caso utilize essa função da unit *Dialogs*, a mensagem será disparada apenas no servidor, travando a aplicação no cliente até a mensagem ser liberada. Para contornar essa limitação será criado um formulário de mensagem. Para isso, vá no menu File>New>Other e na aba Raudus adicione um RaForm. No novo formulário adicione um TRaLabel, um TRaPanel e dentro do painel adicione um TRaButton.

O formulário de mensagem deve ficar como o da **Figura 6**. Para isso, altere as propriedades de acordo com o especificado na **Tabela 1**.

No evento OnClick do btnOK implemente o comando Close. Em seguida, crie o procedure RaShowMessage(psMsg : String) fora do escopo da classe do formulário e logo depois da diretiva de criação de variáveis globais implemente o código da **Listagem 3**. Salve essa unit com o nome de *uMsg*.

Componente	Propriedade	Novo Valor
Formulário	Name	FrmMsg;
	BorderIcons	[];
	Caption	WebApplication com Raudus;
	Color	clWhite
	Position	poScreenCenter
TRaLabel	Name	lblMSG
	Color	clWhite
	Font.Style	[fsBold]
	Caption	Minha mensagem...
TRaPanel	Align	alBottom
	UI	cupertino
TRaButton	Name	btnOK
	Left	215
	Top	8
	Caption	Ok
	ModalResult	mrOK
	UI	cupertino

Tabela 1. Propriedades dos componentes do formulário de mensagem

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Listagem 3. Método RaShowMessage

```
01 procedure RaShowMessage(psMsg : String);
02 Begin
03   FrmMsg.Position := poScreenCenter;
04   FrmMsg.lblMSG.Caption := psMsg;
05   FrmMsg.ShowModalNonBlocking;
06 End;
```

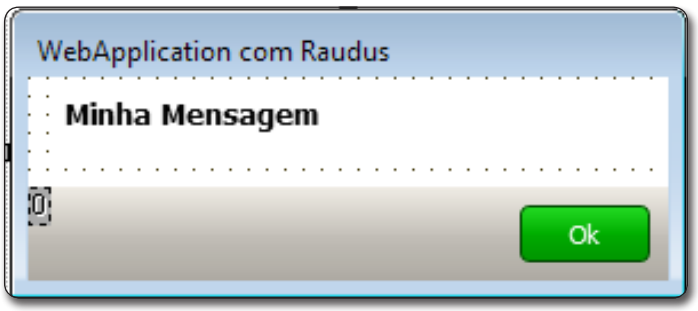


Figura 6. Formulário para simular ShowMessage

Esse método será responsável por manter o formulário de mensagem sempre posicionado no centro, graças à configuração da linha 3. Na linha 4 é passado o valor recebido no parâmetro para o Caption do label do formulário e na linha 5 é apresentado o formulário em show modal. Com isso é possível simular o comportamento do ShowMessage.

Na sequência, retorne ao formulário principal adicione a uses a unit *uMsg* e no evento OnClick do btnOla adicione o seguinte método:

```
RaShowMessage('Ola ' + edtNome.Text);
```

Salve e compile o projeto e a aplicação será executada na forma de prompt de comandos. Para abri-la no navegador digite o endereço *http://localhost:<porta>/* onde <porta> é a porta de acesso ao serviço, que pode ser definida utilizando o comando *Application.Config.Port:=<porta desejada>*. No final desse processo tem-se um resultado semelhante ao desenvolvimento com Intraweb e UniGUI.

O Raudus não suporta alteração do visual com CSS, pois tem um número bem inferior de componentes e o código HTML gerado por ele para interpretação pelo browser é bem mais "sujo" que o do UniGUI. Devido a não retornar as classes de cada elemento, a implementação de CSS é dificultada de uma forma geral para um grupo de elementos. Como pode ser observado, ao utilizar o recurso de desenvolvedor do browser, os estilos são definidos diretamente com o uso do atributo *style* em cada elemento.

Ao clicar no botão Olá e exibir a mensagem percebemos que o formulário não tem uma boa aparência, e para contornar isso o Raudus tem uma unit chamada de *RxTheme*, que pode ser encontrada na pasta de instalação dos demos dentro da pasta *RxTheme*. Essa unit contém a classe *RxTheme*, que permite que se aplique temas a um determinado formulário. Para implementar esse recurso adicione a unit ao projeto e na cláusula *uses* do formulário de mensagem adicione a unit *RxTheme*. No Create do formulário implemente o seguinte comando:

```
TRxTheme.Create(Self).ForceMobile := True;
```

Salve e compile o projeto, que ao ser executado deve ficar como na **Figura 7**. Para implementar em outros formulários basta seguir os mesmos passos.

O método *ForceMobile* melhora o visual para um padrão de view mobile que pode ser implementado também para view web e fica com um visual bastante agradável.

Figura 7. RaShowMessage utilizando ForceMobile

O Raudus é indicado para pequenos projetos com número limitado de recursos e onde o visual não vai sair muito do padrão por ele determinado. Dentro do diretório de instalação há uma pasta contendo uma série de exemplos de implementações utilizando-o.

DataSnap

DataSnap é uma tecnologia baseada inicialmente no DCOM, que permite a criação de aplicações multi-tier. Com o passar dos anos, o DataSnap sofreu modificações e foi introduzida uma nova arquitetura não baseada mais sobre o DCOM, mas sim no Indy, permitindo a comunicação sobre protocolos HTTP e HTTPS.

Com o lançamento do Delphi 2010 a arquitetura DataSnap ganhou novas implementações, permitindo a criação de serviços REST (*Representational State Transfer*) chamado de DataSnap REST

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

e passou a trafegar os dados com formato JSON. O REST é uma arquitetura *client-server* utilizada na Word Wide Web e pressupõe que o servidor é capaz de servir solicitações dos tipos GET, POST, PUT, DELETE, dentro outros menos utilizados. Essa arquitetura assume que cada método do servidor pode ser invocado com uma dessas operações através de um mecanismo de despacho que assume um mapeamento entre o caminho URI (*Uniform Resource Identifier*), nome do método e parâmetros. Isto permite, com apenas um método, manipular um ou mais elementos no servidor sem a necessidade de um método para cada ação.

O DataSnap permite que o serviço seja consumido por qualquer outra linguagem, garantindo maior escalabilidade aos projetos. Um projeto DataSnap REST permite que requisições possam ser respondidas com HTML. Além disso, ele cria uma biblioteca de JavaScript para acesso aos seus recursos, permitindo que sejam desenvolvidos projetos web no Delphi, isolando ou não a regra de negócio das views. Podem ser criados projetos com Dephi no *back-end*, assim como PHP ou JSP. Para esse tipo de desenvolvimento se faz necessário conhecimento de HTML, JavaScript e CSS e o projeto criado utilizando essa técnica será um projeto misto utilizando o melhor de cada linguagem

Para criar um projeto utilizando essa técnica, acesse o menu File>New>Other e na aba DataSnap REST Application selecione uma das opções:

- para aplicações Windows utilize a opção Stand-alone VCL Application,
- para hospedagem no ISS ou Apache utilize o ISAPI Dynamic Link Library.

Na próxima tela informe a porta desejada e clique em Next. Em Server Methods Class marque as duas opções e clique em Next novamente, fazendo assim com que seja gerada automaticamente uma classe *ServerMethods* com alguns métodos de exemplo (*Sample Methods*). Escolha na sequência a opção TComponent, classe da qual a *ServerMethods* irá herdar e que traz as funcionalidades básicas necessárias para esse exemplo. Clique em Next e, na próxima janela (em Project Location) defina a pasta de output dos arquivos gerados pelo projeto. Para finalizar, clique em Finish.

Um projeto DataSnap REST irá gerar uma série de arquivos HTML, CSS, JS, imagens e PAS. A estrutura do projeto será semelhante à daqueles desenvolvidos com frameworks web como Ruby On Rails, Play Framework, entre outros. Na **Figura 8** está a representação da estrutura criada pelo projeto.

Como vemos são gerados três arquivos .pas:

- *ServerMethodsUnit1*.pas: Serve como repositório para os métodos que serão consumidos. Como definido no wizard, a classe herda de TComponent e já possui dois métodos de exemplo;
- *FormUnit1*.pas: É responsável por prover a interface principal da aplicação, nele existem alguns controles que são adicionados pelo Wizard e alguns métodos implementados;
- *WebModuleUnit1*.pas: Serve de repositório para os componentes que transformam a aplicação em um servidor REST DataSnap. Nesse formulário o wizard insere vários componentes que serão responsáveis por receber, tratar e responder as requisições. Além dos componentes são inseridos alguns métodos, dentre os quais vale destacar o *WebModuleDefaultAction*, que irá trabalhar com as rotas. Nesse método pode-se capturar uma requisição e repassar ela para um contexto de resposta desejado. Na **Tabela 2** será descrito um pouco de cada um dos componentes presentes nessa unit.

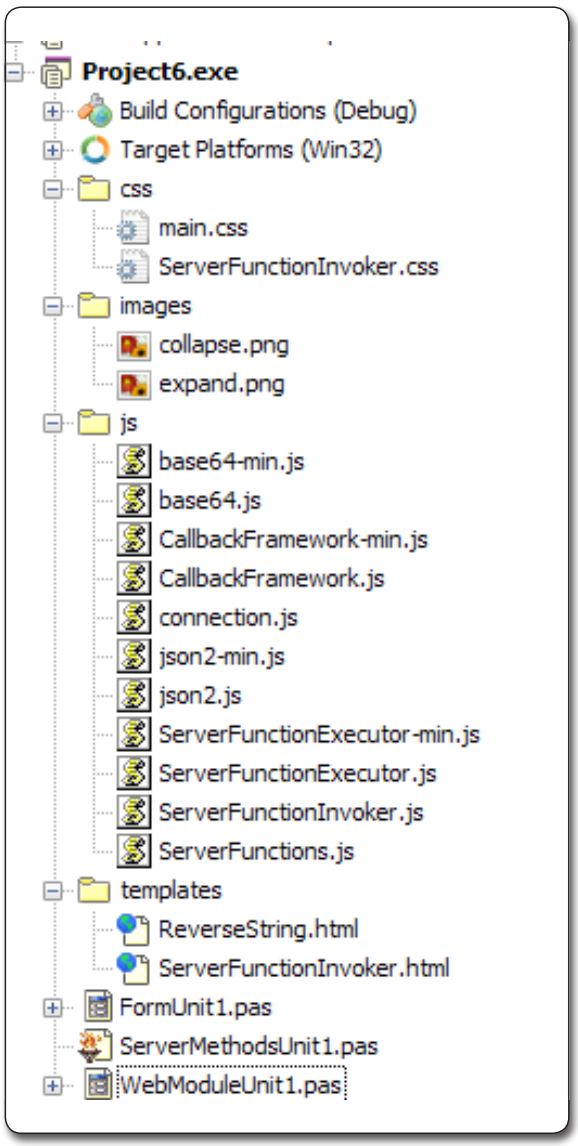


Figura 8. Estrutura do projeto DataSnap REST

Nota

O método ServerFunctionInvokerHTMLTag é executado pelo evento OnHTMLTag do componente TPageProcedure e valida a existência do elemento com o sinal de cerquilha (#). Com isso podemos processar o HTML no Delphi antes de ser enviado para o cliente.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Componente	Descrição
DSServer1	Responsável por transformar uma aplicação em um servidor.
DSServerClass1	Responsável por "expor" uma classe para que a mesma possa ter seus métodos consumidos pela aplicação cliente. Somente classes que descendem de TPersistent e possuam as diretivas {$MethodInfo ON} e {$MethodInfo OFF} em sua estrutura poderão ser consumidas.
DSHTTPWebDispatcher1	Responsável por receber as requisições feitas ao servidor DataSnap.
DSServerMetaDataProvider1	Responsável pelo intermédio entre um DSServer e um ProxyGeneration que provê um MetaDataProvider para geração dos Proxys.
DSProxyGenerator1	Responsável por gerar as classes proxys para se conectar e consumir um servidor de aplicação.
WebFileDispatcher1	Responsável por gerencia a requisição aos arquivos e diretórios que compõe o servidor.
ReverseString ServerFunctionInvoker	Componente do tipo Page Producer utilizado para trabalhar com templates HTML.

Tabela 2. Componentes do WebModules

Para prosseguir com este exemplo, nessa etapa deverão ser removidos todos os arquivos CSS, HTML e imagens da pasta templates, pois não serão utilizados aqui. Em seguida, no *WebModule* remova os componentes *ServerFunctionInvoker* e *ReverseString*. Já no método *ServerFunctionInvokerHTMLTag* remova o excesso de IFs, deixando apenas até a condição do "classname", assim esse método deve ficar igual a **Listagem 4**.

Listagem 4. Método ServerFunctionInvokerHTMLTag

```
01 procedure TWebModule1.ServerFunctionInvokerHTMLTag(Sender: TObject; Tag:
02 TTag;
03   const TagString: string; TagParams: TStrings; var ReplaceText: string);
04 begin
05   if SameText(TagString, 'urlpath') then
06     ReplaceText := string(Request.InternalScriptName)
07   else if SameText(TagString, 'port') then
08     ReplaceText := IntToStr(Request.ServerPort)
09   else if SameText(TagString, 'host') then
10     ReplaceText := string(Request.Host)
11   else if SameText(TagString, 'classname') then
12     ReplaceText := TServerMethods.ClassName
13 end;
```

Na sequência, no *ServerMethods* remova todos os métodos e depois clique com o botão direito sobre o projeto e acesse a opção Add New > Other. Na aba Web Documents selecione a opção HTML Page: salve-a dentro da pasta templates com nome de *index.html*. Edite o HTML com o próprio Delphi ou um editor de sua preferência, alterando o *body* para que fique como o da **Listagem 5**.

Nessa listagem foi adicionado ao HTML um título do tipo H3, um input para receber um texto do usuário e um botão para lançar o evento. Também foi adicionado o atributo *style* para colocar os elementos separados e alterar seu tamanho. Na linha 3 foi informado o id do input para que os dados desse elemento possam ser capturados utilizando a função JavaScript *getElementById*. Já na linha 4 adicionado ao botão um evento de OnClick, disparando o método *onOlaNome*. Esse método será adicionado a seguir nos scripts e será responsável por fazer a requisição ao servidor e exibir a mensagem na página. No body foi adicionado ao evento de OnLoad a chamada ao método *onLoad*, que será responsável por fazer os processamentos necessários ao carregar a página.

Listagem 5. Código body do index.html

```
01 <body onload="onLoad()">
02   <h3>Insira seu Nome</h3>
03   <input id="edtNome" type="text" style="display:block;width:150px"/>
04   <input type="button" onclick="onOlaNome()" value="Ola"
       style="display:block;margin-top:5px;width:150px"/>
05
06 </body>
```

Salve as alterações e visualize o a página. Neste momento será possível ver que a página criada ficou semelhante aos das aplicações com Intraweb, UniGUI e Raudus.

Para que o servidor disponibilize essa página para o cliente em uma requisição adicione um componente *TPageProducer* no *WebModule* e altere o nome desse componente para *Index*. Na propriedade *HTMLFile* do componente *Index* coloque o caminho da página *index.html* (*templates\Index.html*). Em seguida, no componente *Index* ligue o evento *OnHTMLTag* ao método *ServerFunctionInvokerHTMLTag*.

No método *WebModuleDefaultAction* do *WebModule* altere no bloco da primeira condição o *ReverseString* para *index*, ficando assim:

```
Response.Content := Index.Content;
```

Com isso, as requisições feitas diretamente ao servidor sem um contexto serão respondidas com a página index. Agora já se pode salvar e compilar o projeto, o que fará com que seja aberta a aplicação servidora Stand Alone. Clique em *Start* para

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

iniciar o serviço e *Open Browser* para visualizar o resultado. Será exibida uma página web, onde já se pode adicionar informação ao edit, contudo, sem a existência de funcionalidade ao clicar no botão

Para o evento do botão poderíamos apenas adicionar um código JavaScript para exibir uma mensagem com a função *alert*. Dessa forma, no entanto, não existiria nenhuma interação com o servidor DataSnap. Por questões didáticas veremos como o JavaScript pode fazer uma requisição ao servidor, passando como parâmetro o valor do input edtNome. Para isso, no *ServerMethods* adicione a function *Ola(Value: string): string* na classe e em sua implementação adicione a seguinte linha:

```
Result := 'Ola ' + Value;
```

Para acessar e consumir esse método através do JavaScript será utilizada a biblioteca de comunicação criada pelo Delphi que se encontra dentro do projeto na pasta *js*. Então edite o *index.html*, adicionando os scripts JavaScript da **Listagem 6**. Essa lista de scripts é responsável por se comunicar com o servidor e chamar o método desejado. E depois dela adicione o código da **Listagem 7**.

Listagem 6. Lista de scripts JS da biblioteca para comunicação com o servidor DataSnap

```
01  <script src="js/connection.js"></script>
02  <script src="js/serverfunctionexecutor.js"></script>
03  <script src="js/serverfunctions.js"></script>
04  <script src="js/ServerFunctionInvoker.js"></script>
```

Listagem 7. Script JS para conexão e chamada do método no servidor

```
01  <script>
02    function onLoad(){
03      setConnection('<#host>','<#port>','<#urlpath>');
04    }
05
06    function serverMethods(){
07     return new <#classname>(connectionInfo);
08    }
09
10    function onOlaNome(){
11      var sNome = document.getElementById('edtNome').value;
12      alert(serverMethods().Ola(sNome).result);
13    }
14  </script>
```

Na linha 3 da **Listagem 7** é chamando o método *setConnection*, que é responsável por definir os dados de conexão com o servidor a partir do host, da porta e do caminho do URL informados como parâmetro. Na linha 7 a função *ServerMethods* instancia e retorna um novo objeto da classe que foi gerada automaticamente, a partir da qual serão invocados os métodos do lado servidor. Note que nas duas funções foram utilizadas tags com o símbolo de cerquilha (#) precedendo seu nome. Isso faz com que essas tags sejam substituídas durante a execução pelos respectivos valores configurados na criação do projeto.

O método *serverMethods* retorna um objeto da classe *TServerMetods* do JavaScript. No método *onOlaNome* é pego o valor do *edtNome* através da função *getElementById* e repassado para a variável *sNome*. Na linha 12 é utilizado o método similar ao *ShowMessage* do Delphi no JavaScript, o *alert*. Para essa função é passado como parâmetro o resultado da execução do método *Ola* no servidor DataSnap.

Para adicionar o CSS à aplicação, como nas anteriores, crie um arquivo CSS, implemente-o como na **Listagem 1** e salve na pasta *css* do projeto com nome de *WebApplication.css*. Em seguida, no *head* do *index.html* adicione a seguinte linha:

```
<link rel="stylesheet" type="text/css" href="css/WebApplication.css"></link>
```

Após salvar e compilar o projeto, veremos a na execução da aplicação um resultado idêntico ao das aplicações desenvolvidas anteriormente.

O desenvolvimento de aplicações web utilizando essa técnica é bem simples, contudo exige o conhecimento de linguagens *front-end* utilizadas na web. Por outro lado, isso dá mais liberdade e flexibilidade durante o desenvolvimento da aplicação, e permite que o projeto possa ser melhor dividido em camadas, separando bem a regra de negócio da apresentação. Isso facilita, por exemplo, a utilização do modelo MVC (*Model–View–Controller*).

Apesar de a utilização do DataSnap não permitir um desenvolvimento tão semelhante ao de aplicações desktop, como nas seções anteriores, essa opção facilita a divisão de trabalho entre a equipe, delegado aos profissionais especializados no *front-end* e no *back-end* suas funções específicas. Além disso, a aplicação de técnicas comumente usadas na web é facilitada graças ao acesso simplificado aos componentes *client-side* da aplicação (HTML, CSS, JavaScript).

Cada uma das tecnologias exploradas neste artigo tem suas vantagens e desvantagem e podem ser melhor utilizadas em contextos específicos. A Intraweb se aplica muito bem em aplicações para as quais se disponibiliza de um profissional de web design, devido ao seu visual padrão não ser tão elegante e não disponibilizar de recursos de temas e skins. Essa biblioteca não necessita de desenvolvedores com nenhum grau de conhecimento de tecnologias web, porém esse conhecimento certamente agrega grande valor ao projeto. Seu número de componentes é satisfatório para o desenvolvimento de aplicações de grande porte e aplicar CSS ao projeto é simples.

O UniGUI, por sua vez, é bastante adequado para projetos nos quais não se tem grande necessidade de um designer para construir interfaces gráficas mais elaboradas, pois seu visual padrão, com recursos de temas e skin, é bem satisfatório. Assim como a Intraweb não necessita de profissionais com conhecimento em tecnologias web, isso pode auxiliar na construção de páginas com visual mais elegante. O UniGUI permite um

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

REVISTAS DIGITAIS

Imagine poder ter acesso a todas as edições das revistas Java Magazine, .NET Magazine, SQL Magazine, Mobile Magazine, Engenharia de Software, ClubeDelphi e Easy Java.

São mais de 4.000 artigos publicados!

Uma verdadeira biblioteca online perfeita para seus estudos!

Para mais informações :
http://www.devmedia.com.br/mvp

DEVMEDIA

desenvolvimento misto entre linguagens, fazendo com que muitos recursos sejam desenvolvidos com a utilização de JavaScript e jQuery, se necessário. Esse framework também oferece maior interação entre Delphi e eventos JavaScript e tem uma boa gama de componentes, permitindo o desenvolvimento de aplicações de grande porte sem grandes problemas. A implementação de CSS não é tão simples quanto com a Intraweb devido à forma como são estruturados os elementos, mas ainda assim é possível implementar ótimas melhorias visuais.

O Raudus é mais indicado para projetos mais simples, devido ao número limitado de componentes e recurso, porém não exige um designer e profissionais com conhecimento de tecnologias web, pois já disponibiliza de temas para recursos visuais A implementação de CSS no Raudus ainda não é possível até a versão 0.8.7.

O DataSnap é a opção mais adequada para projetos onde existem profissionais com conhecimentos das tecnologias web e pode ser utilizado na construção de serviços REST, para disponibilização de funcionalidades para outras aplicações clientes, que inclusive podem ser construídas com outras linguagens de programação, bastando que haja suporte a requisições HTTP. Esse framework é a opção mais adequada quando escalabilidade e design são fatores decisivos, pois sua estrutura permite a construção de arquiteturas maduras, utilizando os principais padrões de projeto, além de adequada separação de responsabilidades entre os componentes da aplicação.

Autor

Gutierry Antonio
gutierrydsn@hotmail.com

Entusiasta de Big Data e Data Warehouses, atua como Software Engineer e DBA MySQL. É graduado em Sistemas da Informação e Análise de Sistemas, e atualmente cursa especialização em Engenharia de Sistemas. Possui experiência em desenvolvimento com Delphi e Ruby on Rails com bancos de dados Firebird e MySQL, além de conhecimento de Java, C# e C++ com SQL Server e Oracle, bem como de AWS.

Links:

UniGUI – Demos
http://www.unigui.com/demo

UniGUI – Fórum do oficial do UniGUI
http://forums.unigui.com/

Sencha – Sencha extjs
https://www.sencha.com/products/extjs/#overview

Raudus - Site Oficial
http://www.raudus.com/

Embarcadero - REST Servers in Delphi XE Using DataSnap
https://www.embarcadero.com/images/dm/technical-papers/rest-servers-in-delphi-xe-using-datasnap.pdf

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Manipulando PDF com GhostScript

Aprenda a ler, criar e comprimir PDF no Delphi

Ao trabalhar com leitura e gravação de arquivos em sistemas, é comum o surgimento de alguns problemas, principalmente relacionados ao armazenamento e, em muitos casos, também com o tempo que esse arquivo leva para trafegar na rede. Os usuários, ao inserirem um documento na aplicação, não estão preocupados com o quanto esse documento irá ocupar em espaço no disco, na verdade muitos deles nem têm essa noção. Cabe ao desenvolvedor entender, identificar e resolver esses problemas, não podendo, muitas vezes, simplesmente gravar em disco diretamente ou em um banco de dados um arquivo inserido pelo usuário, o que pode causar no futuro problemas como excesso de espaço utilizado em disco e perda de desempenho.

Para solucionar, ou ao menos reduzir esse tipo de problema, vamos utilizar uma ferramenta chamada GhostScript, que pode ser integrada a aplicações Delphi e permite, além de comprimir arquivos PDF, reduzindo seu tamanho em disco, extrair dados desse tipo de arquivo, bem como gerar PDFs a partir de uma extensão diferente.

Problemas com arquivos grandes

Quando lidamos com arquivos grandes em nossas aplicações, as principais dificuldades surgem quando se torna necessário transmitir esses arquivos através da internet, seja por e-mail ou via webservice, por exemplo. Em situações mais críticas, pode haver ainda limitações de largura de banda disponível e tráfego máximo permitido por período, muito comum em pacotes de hospedagem de serviços web.

Outra grande dificuldade diz respeito ao espaço de armazenamento físico utilizado e disponível, que tende a ser consumido rapidamente dependendo da frequência com que arquivos são armazenados. Em alguns casos há a necessidade de efetuar melhorias de hardware para suportar o grande volume de dados, situação que pode deixar a aplicação fora de funcionamento por algum tempo, dependendo da infraestrutura da empresa.

Fique por Dentro

Lidar com PDF em aplicações Delphi é uma tarefa bastante comum, seja para armazenamento, leitura ou geração de arquivos para posterior envio por e-mail, por exemplo. Nessas situações é fundamental que o arquivo tenha o menor tamanho possível, de forma que se possa economizar espaço em banco de dados e reduzir o tráfego de rede. O GhostScript, tema central desse artigo, é importante porque além de prover a funcionalidade de compressão de arquivos PDF, também permite extrair dados de um arquivo e gerar outros a partir de extensões diferentes.

Quando esses arquivos são armazenados em banco de dados começam a surgir problemas de desempenho nas consultas e principalmente nas gravações desses registros, além de afetar diretamente a realização dos backups, uma vez que existirá um gasto maior de tempo e espaço para cópia dos mesmos.

Conhecendo o PDF

O PDF (*Portable Document Format*) é um formato de arquivo desenvolvido pela Adobe Systems com o objetivo representar documentos de forma independente do aplicativo, do hardware e do sistema operacional usados para criá-los. Um arquivo PDF pode descrever documentos que contenham texto, gráficos e imagens em um formato compatível com a maior parte dos sistemas operacionais e que podem ser visualizados em diversas aplicações, inclusive browsers.

É possível gerar arquivos em PDF a partir de vários formatos de documentos, como ODF *(LibreOffice)* ou DOC (*Microsoft Word*), além de imagens, como JEG e PNG. No entanto, a qualidade do documento gerado, no que se refere à exibição do conteúdo, pode variar de acordo com o formato do arquivo matriz a partir do qual o PDF foi criado. Portanto, a escolha do formato mais adequado pode ser um esforço válido, principalmente quando se trata de PDFs que contêm informações institucionais a serem entregues a gestores ou clientes e que poderão ser visualizados em dispositivos de diferentes resoluções.

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Sobre o GhostScript

O Ghostscript foi escrito originalmente como um processador de imagem raster (*RIP*) e pode ser utilizado também como conversor de formato de arquivos, como o PostScript para PDF (*GhostPDF*), ou como uma "impressora virtual", suportando PDF/A-1 e PDF/A-2 (**BOX 1**). O GhostScript pode ser utilizado a partir do *Prompt de Comandos*, com chamadas ao seu executável e passagens de parâmetros ou com a inserção de parâmetro em sua própria interface. Ele também disponibiliza uma API (nesse caso uma DLL) que podemos importar para os sistemas e efetuar chamadas diretamente às suas funções, sem a necessidade de instalação de novos recursos.

BOX 1. Padrões PDF/A-1 e PDF/A-2

O padrão PDF/A, também conhecido como ISO 19005-1, é um padrão para manter informações armazenadas em documentos eletrônicos por algum período de tempo. Basicamente cobre aspectos referentes a reprodução, armazenamento e dependência de hardware e software. Sua versão mais recente (A-2) oferece mais suporte a novas tecnologias e se apoia na versão 1.7 do PDF.

Escrito originalmente por L. Peter Deutsch para o Projeto GNU, o GhostScript foi liberado sob a *GNU General Public License* em 1986. Atualmente a ferramenta é propriedade de Artifex Software e mantida por seus funcionários e pela comunidade de utilizadores em todo o mundo.

Por que utilizar

Entre as principais ferramentas de compressão de arquivos PDF existentes no mercado atualmente, podemos observar que das seis ferramentas mais utilizadas (CutePDF Writer, doPDF, PrimoPDF, PDF Printer Community, FreePDF e PDF Creator), cinco utilizam o GhostScript. Essas ferramentas, na sua maioria, não criaram nenhuma funcionalidade extra especial, apenas utilizam a API do GhostScript e disponibilizam uma interface amigável para o usuário final. O GhostScript não só trabalha com o formato PDF, mas também com PCL (*Printer Command Language*), XPS (*XML Paper Specification*) e SVG (*Scalable Vector Graphics*). Neste artigo, porém, veremos apenas a parte do PostScript, que é a biblioteca responsável pela manipulação de PDF. Ela conta com uma boa documentação que pode ser consultada para que se possa aproveitar ao máximo suas funcionalidades (ver seção **Links**).

O que é PostScript

PostScript é uma linguagem de programação especializada para visualização de informações, ou uma linguagem de descrição de páginas, originalmente criada para impressão e posteriormente modificada para o uso com monitores (*display PostScript*).

A linguagem fornece uma máquina de pilha e comandos específicos para o desenho de letras e figuras, incluindo comandos de traçado e formas de representação de imagens. Foi desenvolvida pela Adobe, tendo como inspiração a InterPress, uma linguagem desenvolvida na Xerox.

A PostScript é considerada uma linguagem de programação Turing completa, ou seja, é possível implementar qualquer algoritmo computacional respeitando-se os limites de memória de dados. Tipicamente os programas PostScript não são produzidos por humanos, mas sim por outros programas de computador. É possível, entretanto, produzir gráficos ou cálculos através da programação manual diretamente em PostScript. Trata-se de uma linguagem interpretada, orientada por pilha de dados, com algumas características semelhantes à linguagem Forth, incorporando um grande dinamismo de tipos de dados, com estrutura desses semelhantes à linguagem Lisp. A linguagem utiliza a sintaxe da notação reversa polonesa, onde o uso de parênteses não é necessário.

A programação manual em PostScript exige alguma prática, considerando que o layout da pilha de dados deve ser planejado pelo programador. A maioria dos operadores (funções) extrai os seus argumentos da pilha de dados e, quando é o caso, devolve o resultado na pilha. Dados literais ocasionam o efeito de colocar uma cópia de si mesmo na pilha de dados. Estruturas de dados complexas podem ser construídas através de arrays e entradas de dicionário.

Problemas ao trabalhar com PDF com GhostScript

Ocasionalmente, você pode tentar ler ou imprimir um arquivo PDF que o GhostScript não reconhece como tal, embora o mesmo arquivo possa ser aberto e interpretado por um visualizador Adobe Acrobat Reader, por exemplo. Em muitos casos, isso ocorre quando o PDF é gerado incorretamente. O Acrobat Reader tende a contornar algumas falhas nos arquivos PDF inválidos, já o GhostScript espera que os arquivos tratados estejam em conformidade com a norma do formato. Por exemplo, apesar da regra para arquivos PDF válidos ser começar com %PDF, o Acrobat digitaliza os primeiros 1000 bytes para esta cadeia, e ignorar qualquer lixo anterior.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

No passado, a política do Ghostscript era a de simplesmente falhar com uma mensagem de erro quando confrontado com estes arquivos. Esta política, sem dúvida, encorajava geradores de PDF a serem mais cuidadosos. No entanto, este comportamento não é muito amigável para as pessoas que querem apenas usar o GhostScript para visualizar ou imprimir arquivos PDF. A nova política é tentar tornar PDFs corrompidos utilizáveis imprimindo um aviso, de modo que GhostScript continua útil quando se tenta abrir arquivos inválidos.

Um outro problema percebido ao se trabalhar com PostScript é na geração do PDFs. Algumas fontes não são reconhecidas e o documento gerado acaba não refletindo 100% do original. Uma solução é a inserção dessas fontes na hora da geração do PDF, porém isso é muito trabalhoso e se perde a compactação das fontes, gerando um arquivo final muito grande.

Obter e instalar o Ghostscript

A versão não-comercial do GhostScript é distribuída sob licença *GNU Affero GPL*, que permite uso e distribuição livres e gratuitos (*open source*).

A distribuição inclui pacotes para Windows prontos para uso, com instaladores distintos para 32 e 64 bits, que podem ser obtidos na página oficial do GhostScript (seção **Links**).

Basta executar o instalador Windows e seguir o passo-a-passo do assistente. O programa instalador tem nome no formato gs<versão>w32.exe. No local de instalação no Windows, normalmente em C:\Arquivos de programas\gs\, é criada uma pasta gs<versão>\, de acordo com a versão do GhostScript instalada. A pasta bin contém um arquivo com nome gswin32c.exe e outro gswin32.exe, que é o aplicativo de interação, onde iremos adicionar os comandos para o GhostScript processar. Juntamente aos arquivos EXE temos a DLL que iremos utilizar para integração com o Delphi, a gsdll32.dll.

Executando o Ghostscript

O GhostScript pode ser executado de duas formas: através do *Prompt de Comandos* ou com duplo clique sobre o arquivo gswin32 .exe. Ao executar pelo prompt os parâmetros devem ser passados junto à sua chamada, já pelo executável é aberta uma tela onde podemos passar os parâmetros. Para executar através do prompt devemos digitar o seguinte comando: "gswin32 [options] {filename 1}", para isso deve existir um *Path* configurado para a pasta do GhostScript ou podemos abrir o terminal diretamente pela pasta *bin* ou colocarmos o caminho mais o nome do executável e os parâmetros no prompt.

Vamos agora começar a usar pequenas funcionalidades dessa biblioteca. Com o *Prompt de Comandos* aberto vamos digitar a seguinte linha:

```
gswin32 -dSAFER -dBATCH <caminho de um arquivo pdf>
```

Esse comando vai fazer com que o arquivo seja apresentado em uma tela do próprio GhostScript, com isso já podemos eliminar a necessidade do cliente ter um leitor de PDF instalado em sua máquina ou efetuar chamadas a executáveis de terceiros para apresentar PDF.

Podemos ainda definir quais páginas queremos recuperar, para isso utilizamos o parâmetro *-dFirstPage* para determinar a página inicial e o *-dLastPage* para determinar a página final. Caso desejemos recuperar apenas a página 5 e 6 de um documento, por exemplo, utilizaremos a seguinte linha de comando:

```
gswin32 -dSAFER -dBATCH -dFirstPage=4 -dLastPage=4 <caminho de um arquivo pdf>
```

Além de abrir um PDF, podemos gerar um arquivo a partir de um segundo, seja ele de mesma ou de outra extensão, para isso utilizando o comando:

```
gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=jpeg -sOutputFile=figure.jpg
<meuArquivo.PDF>
```

Esse comando utiliza como device de output o device jpeg e gera um arquivo JPG com nome *figure.jpg* no mesmo diretório do GhostScript a partir do *meuArquivo.PDF*. Nessa linha utilizamos três novos comandos: *-dNOPAUSE*, *-sDEVICE* e *-sOutputFile*. Vamos entender esses parâmetros: o *-dNOPAUSE* informa ao GhostScript que no processo não existirá interação com o usuário, ou seja, em casos onde seja preciso pressionar o *Enter* esse comando removerá essa necessidade. O *-sDEVICE* serve para definirmos qual o device ("impressora virtual") será utilizado. Esse device é o que define qual a saída ou como será tratado esse documento, nesse caso utilizamos como device uma saída JPEG. Por último temos o parâmetro *-sOutputFile*, onde informamos o caminho de saída do documento que será gerado a partir do documento original que se encontra informado na frente do arquivo de output. Como não informamos um caminho completo para esse documento, ele será gerado dentro da pasta *bin* do

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

GhostScrip, juntamente do seu executável.

Podemos utilizar essa mesma técnica para gerar um arquivo TXT com o texto do PDF, para isso basta que como device de output utilizemos o device *txtwrite*:

```
gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=txtwrite -sOutputFile=file.txt
MeuPDF.PDF
```

Com apenas uma linha de comando temos a capacidade de extrair os dados de um PDF para um arquivo de texto e trabalhamos sobre esse arquivo como preferirmos. Uma aplicação prática para isso seria a extração do texto do PDF e armazenamento dessa informação em banco de dados para permitir ao cliente efetuar consultas posteriormente. Nesse mesmo cenário podemos também aplicar validações sobre o conteúdo do PDF antes de inseri-lo no sistema, por exemplo. A aplicação de novas funcionalidades depende apenas das necessidades do projeto.

Para sabermos quais devices estão disponíveis, podemos digitar no prompt a seguinte linha de comando:

```
gswin32 -h ou gswin32 -?
```

Além de apresentar um help, será também exibido logo abaixo um trecho "*Available devices*", onde estarão descritos todos os devices disponíveis. Veremos que existem devices para trabalharmos com inúmeros tipos de arquivos (JPEG, BMP, PNG, TIFF e PDF).

Para obtermos suporte sobre suas funções podemos utilizar a seguinte linha de comando:

```
gswin32 -h ou gswin32 -?
```

Para solucionar o problema com fontes descrito anteriormente podemos utilizar o parâmetro *-sFONTPATH=* para informar o caminho da fonte que não foi reconhecida.

Caso o arquivo com o qual estejamos lidando esteja com alguma senha aplicada, podemos utilizar do parâmetro para *sPDFPassword* para definir a senha do usuário a ser utilizada na decodificação do PDF criptografado.

Comprimindo arquivos

Para geramos arquivos PDF menores com o GhostScript é muito simples, basta criarmos um arquivo PDF a partir de um outro já existente utilizando o device *pdfwrite*:

```
gs -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -sOutputFile=PDFComprimido.pdf
PDFOriginal.pdf
```

Essa linha de código pega o arquivo *PDFOriginal.pdf* e gera um novo com nome de *PDFComprimido.pdf* na saída a partir do device *pdfwrite*. O GhostScript já tem uma inteligência capaz de reduzir sem a inserção de nenhum novo parâmetro, porém podemos melhorar essa compressão com a inserção de alguns argumentos para tratamentos específicos.

Nota

A compressão pelo GhostScript é feita sempre que possível, em casos em que o arquivo original já se encontrar comprimido, não será possível efetuar a compressão.

Para que possamos observar o processo compressão na prática e avaliar os resultados, vamos trabalhar com os três arquivos que são criados juntamente com a instalação do GhostScript: *annots.pdf*, *text_graph_image_cmyk_rgb.pdf* e *text_graphic_image.pdf* que se encontram na subpasta *examples* no diretório de instalação. E para facilitar vamos renomear os arquivos para *File1.pdf*, *File2.pdf* e *File3.pdf* respectivamente, além copiá-los para a pasta *bin*, de forma que que possamos trabalhar sem a necessidade de informar o caminho completo dos arquivos, reduzindo assim nossas linhas de comando.

Agora vamos gerar um arquivo comprimido para cada um desses PDFs, bastando para isso executar os comandos da **Listagem 1** no *Prompt de Comandos*.

Para fins de comparação utilizaremos a **Tabela 1**, onde podemos observar o tamanho original de cada um dos arquivos e o tamanho final dos arquivos gerados a partir de sua compressão.

Listagem 1. Comandos para gerar novos PDF a partir do GhostScript

```
01 gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -s
   OutputFile=file1_compressed.pdf file1.pdf
02 gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -s
   OutputFile=file2_compressed.pdf file2.pdf
03 gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -s
   OutputFile=file3_compressed.pdf file3.pdf
```

Arquivo	Tamanho Original	Tamanho pos Compressão	Taxa de compressão
File1.pdf	385 KB	352 KB	8%
File2.pdf	342 KB	117 KB	65%
File3.pdf	131 KB	57 KB	56%

Tabela 1. Comparação dos tamanhos dos arquivos após a compressão

Podemos ver que não foi necessário nenhum parâmetro adicional, apenas geramos um novo arquivo utilizando o device *pdfwrite*, com isso já obtivemos uma boa compressão. Para melhorar essa taxa vamos gerar novos arquivos utilizando os comandos da **Listagem 2** no prompt.

Adicionamos apenas o parâmetro *-dPDFSETTINGS* e setamos seu valor para "*/ebook*", com isso o GhostScript gera uma melhoria na taxa de compressão. Na **Tabela 2** podemos analisar o resultado em comparação à compressão sem o parâmetro.

Após a compressão com a utilização do parâmetro *-dPDFSETTINGS* tivemos uma redução ainda maior no tamanho dos

 Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

arquivos finais, e mesmo assim não houve perda de qualidade. O arquivo *File3.pdf* teve a menor diferença na redução do tamanho, diferença que se deve ao nível de compressão já aplicado sobre esse arquivo e o GhostScript não conseguir aplicar ainda mais. É possível forçar a compressão sobre os arquivos, mais isso poderá acarretar em perda de qualidade.

Listagem 2. Comandos para gerar e mehorar a compressão dos PDF a partir do GhostScript

```
01 gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -dPDFSETTINGS=/
    ebook -sOutputFile=file1_compressed.pdf file1.pdf
02 gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -dPDFSETTINGS=/
    ebook -sOutputFile=file2_compressed.pdf file2.pdf
03 gswin32 -dSAFER -dBATCH -dNOPAUSE -sDEVICE=pdfwrite -dPDFSETTINGS=/
    ebook -sOutputFile=file3_compressed.pdf file3.pdf
```

Arquivo	Tamanho Original	Compressão com dPDFSETTINGS	Taxa de compressão
File1.pdf	385 KB	342 KB	11%
File2.pdf	342 KB	85 KB	75%
File3.pdf	131 KB	56 KB	57%

Tabela 2. Comparação de compressão com configuração -dPDFSETTINGS

Parâmetros do GhostScript

Aqui contemplaremos apenas alguns dos parâmetros que são mais importantes para este artigo, visto que o GhostScript possui um número grande de argumentos possíveis. Mais informações sobre os demais parâmetros podem ser obtidas na página Ghost-Script (seção **Links**). A seguir temos a lista dos parâmetros a serem trabalhados nesse artigo:

• **-sDEVICE=<device_out_put>:** Parâmetro utilizado para definir qual dispositivo de saída será utilizado. Em nosso caso utilizaremos o device *pdfwrite* que é nossa "impressora virtual" para gerar PDF para PS, PS para PDF e PDF para PDF. Essa impressora não é apresentada no painel de controle e em nenhum lugar a não ser no próprio GhostScript, sendo um recurso nativo dele.

• **-dCompressFonts=<True_Ou_False>**: Define se iremos utilizar compressão de fontes ou não, recebendo "*True*" ou "*False*". Testes revelaram que o ganho com esse parâmetro é pouco, mais ainda assim seu uso é válido para arquivos com muito texto.

• **-dPDFSETTINGS=/<tipo>:** Parâmetro utilizado para definir um tipo de configuração de saída padrão, podendo ser ela para visualização em vídeo ou impressão. Essa configuração determina muitos padrões automáticos de compressão, garantindo ganhos visíveis, porém ao definir esse parâmetro como "*Screen*" as perdas de qualidade nas imagens são visualmente percebidas, por isso aqui utilizaremos o valor "*ebook*". Esse parâmetro cria arquivos maiores que com "*Screen*" porém com perda visual muito menor ou quase zero.

• **-dDetectDuplicateImages=<True_Ou_False>:** Aqui definimos se iremos utilizar a detectação de imagens duplicadas, com isso gravando apenas uma referência no lugar das demais imagens duplicadas. Em caso de se encontrar imagens duplicadas a diferença no tamanho dos arquivos é considerável.

• **-dQFactor=<fator>:** Para esse parâmetro deveremos utilizar um valor entre 0.1 e 1.0 como fator para compressão, sendo 1.0 sem compressão e 0.1 o maior nível de compressão. Testes com esse parâmetro indicaram que valores até 0.2 em arquivos JPG em tons coloridos quase não geram perda visual na qualidade e a compressão chegou a mais de 90% na maioria dos casos. Por exemplo, arquivos de 15MB tiveram seu tamanho reduzido para 130KB. Porém, em arquivos com tons de cinza ou preto e branco existe uma perda visual muito significativa, em alguns casos impedindo a identificação da imagem.

• **-dJPEGQ=<valor>:** Para esse parâmetro devemos determinar um valor na faixa de 0 a 100, sendo que quanto menor for o valor, menor será a qualidade das imagens JPEG. Apesar das configurações serem "exclusivas" para JPEG, as mesmas se aplicam a outros tipos de imagens, como PNG, GIF, TIF e BMP em alguns casos. Assim como o *-dQFactor* esse parâmetro causa problemas com imagens com tons de cinza e preto e branco.

• **-dDownsample<Tipo>Images=<True_Ou_False>**: Parâmetro utilizado para definir se será aplicada redução de qualidade (*True* ou *False*) para o tipo de imagem (Color, Gray, Mono). Iremos utilizar um parâmetro para cada tipo com valor *True*, ou seja, iremos aplicar compressão e baixa resolução para cada tipo de imagem.

• **-d<tipo>ImageResolution=<Taxa>**: Nesse parâmetro informamos qual será a taxa de resolução aplicada para cada tipo de

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

imagem (Color, Gray, Mono), podendo ser desde 0 até 3000. Com esse parâmetro definiremos para cada tipo de imagem se ocorrerá uma redução ou "melhora" na imagem. Quanto maior o valor, maior será sua resolução.
Em testes foi visto que com perdas visuais zero o valor ideal é de 200, pois acima disso o arquivo tem seu tamanho aumentado e não é percebida diferença visual, principalmente para arquivos de impressão. Com menos de 200 temos melhores resultados na compressão, porém temos perdas perceptíveis, principalmente em casos de arquivos em tons de cinza e monocromático, por isso no tipo deve-se ter uma configuração para cada tipo de imagem.
• **-sOutputFile=<Destino_do_Arquivo>**: Parâmetro utilizado para definir qual o local de saída do arquivo e seu nome. O caminho dos arquivos não deve conter espaços ou caracteres especiais, caso contrário o GhostScript pode gerar um erro por não conseguir identificar o local de destino.

Integração entre Delphi e GhostScript

Antes de iniciar nossa integração, vamos entender como ela é feita. Toda a comunicação do Delphi com o GhostScript é feita através de uma DLL, o que nos permite interagir de forma padronizada com sua API, visto que a DLL deve ser escrita sobre um padrão e todas as linguagens têm a capacidade de interpretar seus recursos. Porém, para que esses recursos fiquem disponíveis, devemos importa-los para nossa aplicação. Para isso devemos escrever uma unit que terá declarados os métodos desejados e farão referência ao método na DLL. Esta serve de "ponte" entre o resto da nossa aplicação e a biblioteca do GhostScript. Essa técnica não é exclusiva do GhostScript e serve para qualquer comunicação com DLL. As DLLs podem ser carregadas de forma dinâmica ou estática. Dinamicamente ela é carregada sempre que temos a necessidade de utilizá-la, e em caso dela não estar disponível e a função não ser executada, nenhum erro será apresentado. Já ao carregar a DLL de forma estática, ela vira uma dependência para o executável e é carregada na sua abertura, independente de se será utilizada ou não, e caso ela não esteja disponível um erro será apresentado e aplicação não irá abrir.

Agora que temos um breve conhecimento do processo de integração, podemos iniciar nossa aplicação. Para isso vamos criar um novo projeto com o nome *DelphiComGhostScript.dproj*. Adicionaremos uma unit que irá importar a DLL do GhostScript, mas como não vamos utilizar toda sua biblioteca vamos importar apenas parte dela. Como podemos observar na **Listagem 3**, declaramos e implementamos quatro métodos externos com chamadas à DLL do GhostScript, que deverá ser distribuída juntamente ao nosso EXE. Os métodos que iremos importar são os métodos para instanciar o GhostScript (gsapi_new_instance), o método responsável por liberar essa instância (gsapi_exit e gsapi_delete_instance) e o método que irá servir como nosso

Listagem 3. Unit para importar biblioteca da dll do GhostScript

```
01 unit uGhostScript;
02
03 interface
04
05 type
06  PPAnsiChar = array of PAnsiChar;
07  {$NODEFINE PPAnsiChar}
08
09   function gsapi_new_instance(var pinstance: Pointer; caller_handle: Pointer)
     : Integer; stdcall;
10   {$EXTERNALSYM gsapi_new_instance}
11
12   procedure gsapi_delete_instance(instance: Pointer); stdcall;
13   {$EXTERNALSYM gsapi_delete_instance}
14
15   function gsapi_init_with_args(instance: Pointer; argc: Integer;
     argv: PPAnsiChar): Integer; stdcall;
16   {$EXTERNALSYM gsapi_init_with_args}
17
18   function gsapi_exit(instance: Pointer): Integer; stdcall;
19   {$EXTERNALSYM gsapi_exit}
20
21 Const gsdll32 = 'gsdll32.dll';
22
23 implementation
24
25 function gsapi_new_instance; external gsdll32 name 'gsapi_new_instance';
26 procedure gsapi_delete_instance; external gsdll32
    name 'gsapi_delete_instance';
27 function gsapi_init_with_args; external gsdll32 name 'gsapi_init_with_args';
28 function gsapi_exit; external gsdll32 name 'gsapi_exit';
29
30 end.
```

"Prompt de Comandos" responsável por passar os parâmetros ao GhostScript (gsapi_init_with_args) e retornar um código do processamento.

O GhostScript fornece outros métodos para importação, vamos dar uma olhada nos principais e quais tarefas são designadas a eles:
• **gsapi_new_instance():** Cria uma nova instância do GhostScript. Essa instância é passada para a maioria das outras funções da API. O *handle* será fornecido para funções de retorno de chamada. O GhostScript permite apenas uma instância de cada vez, qualquer tentativa para criar uma nova resulta em um erro. Se o GhostScript foi compilado com a diretiva *GS_THREADSAFE*, várias instâncias simultâneas são permitidas;
• **gsapi_delete_instance():** Destrói uma instância do GhostScript. Antes de chamar este método o GhostScript deve ter sido terminado utilizando o método **gsapi_exit()**;
• **gsapi_set_stdio():** Se você usar esse método todo *stdio* será redirecionado para as funções de retorno de chamada que você

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

forneceu. Isto seria usado em um ambiente de interface gráfica do usuário onde *stdio* não está disponível, ou onde você deseja processar entradas e saídas do GhostScript;

• **gsapi_set_display_callback():** Define a estrutura de retorno de chamada para o visor do dispositivo. Se o visor do dispositivo é utilizado, então deve ser chamado após **gsapi_new_instance()** e antes **gsapi_init_with_args()**;

• **gsapi_set_arg_encoding():** Define a codificação utilizada para a interpretação de todos os argumentos;

• **gsapi_init_with_args():** Inicializa o interpretador e executa com os parâmetros passados. Esse método "simula" o *Prompt de Comandos*;

• **gsapi_exit():** Sai do interpretador. Este método deve ser chamado para liberar o interpretador se **gsapi_init_with_args()** foi chamado, e pouco antes de **gsapi_delete_instance()**.

Após importamos a biblioteca necessária, nosso próximo passo deve ser criar uma classe que será responsável por efetuar as chamadas e passar os parâmetros para o GhostScript. Nossa classe será nomeada como TCompressaoPDF (**Listagem 4**), criada em uma unit separada que deve ser nomeada como uCompressaoPDF, onde deve ser declarada em sua seção *uses* a unit uGhostScript, que é unit que criamos com as declarações dos métodos importados do GhostScript.

Essa classe é composta de seis propriedades, sendo elas o caminho e nome do documento a ser comprimido (Input), o caminho e nome do arquivo a ser gerado (Output), se deve detectar imagens duplicadas (DetectDuplicateImages) e três parâmetros referentes à resolução de cada tipo de imagem: colorida, tons de cinza e monocromática, respectivamente (ResolutionColor, ResolutionGray e ResolutionMono). Nossa classe também contém dois métodos, sendo o primeiro o Create (construtor, que pode ser visto na **Listagem 5**), onde iremos inserir os valores padrões para nossas propriedades e o segundo o método Executar, onde será implementada a rotina para compressão do PDF, como vemos na **Listagem 6**.

No método Executar foi criada uma constante chamada de *NParams*, que deve ter o valor refletido na quantidade de parâmetros a ser passada ao GhostScript. Esses parâmetros serão inseridos na nossa variável *argv*, sendo ela um array de PChar. Foi declarada também uma variável para armazenar os códigos retornados pelo GhostScript e uma variável que representará a instância do GhostScript.

Após alterar o resultado para *False*, alteramos o tamanho do nosso array *argv* com o valor da constante *NParams*. Em seguida, para cada registro do array passamos um parâmetro e seu respectivo valor, concatenando-os com os valores das propriedades da classe.

Depois de passar todos os parâmetros, dentro de um bloco *try* é feita a chamada ao método de instanciar o GhostScript, que foi importado na uGhostScript. Para esse método passamos no primeiro parâmetro a variável de instância. Caso não ocorra erro ao tentar instanciar, a variável passada receberá a referência da

Listagem 4. Prototype da classe TCompressaoPDF

```
01 TCompressaoPDF = Class
02 private
03 fOutput : String ;
04 fInput : String ;
05 fResolutionColor : Integer;
06 fResolutionGray : Integer;
07 fResolutionMono : Integer;
08 fDetectDuplicateImages : String;
09 public
10 constructor Create;
11
12 Function Executar : Boolean;
13  published
14  Property Output : String read fOutput write fOutput ;
15  Property Input : String read fInput write fInput ;
16  Property ResolutionColor : Integer read fResolutionColor
      write fResolutionColor;
17  Property ResolutionGray : Integer read fResolutionGray
       write fResolutionGray ;
18  Property ResolutionMono : Integer read fResolutionMono
       write fResolutionMono ;
19  Property DetectDuplicateImages : String read fDetectDuplicateImages
       write fDetectDuplicateImages ;
20 End;
```

Listagem 5. Construtor

```
01 constructor TCompressaoPDF.Create;
02 begin
03   inherited;
04   ResolutionColor := 200;
05   ResolutionGray  := 200;
06   ResolutionMono  := 200;
07   DetectDuplicateImages := 'true';
08 end;
```

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

instancia criada pelo GhostScript. Essa função retorna um código, que caso seja diferente de zero, significa que ocorreu algum erro ao tentar levantar essa instância, caso contrário a instância foi criada corretamente. Esse código de erro é passado para a variável *code*. Após a tentativa de instanciar o GhostScript, efetuamos um teste verificando se o valor de *code* é diferente de zero, caso esse valor não seja igual a zero, lançamos uma exceção "Erro ao instanciar" com o código do erro.

Se não houver erro ao instanciar o GhostScript, é então chamado o método *gsapi_init_with_args* da unit uGhostScript, para o qual são passados três parâmetros: a instância que será utilizada, a quantidade de parâmetros e a lista de parâmetros. No primeiro argumento passamos a variável que recebeu a instância do GhostScript, no número de parâmetros passamos a constante *NParams* e a para lista de parâmetros passamos o array de parâmetros o *argv*. O método *gsapi_init_with_args*, assim como o *gsapi_new_instance*, também retorna um código que é armazenado na variável *code*. Caso esse valor seja menor que zero, significa que ocorreu algum problema na geração do PDF, por isso efetuamos essa validação em caso de erro lançamos uma exceção "Erro ao converter" também com o código do erro. Caso não ocorra nenhum erro na chamada do método *gsapi_init_with_args*, é gerado o novo arquivo PDF conforme especificado pelos parâmetros no diretório de saída. Com isso é dado um *result true* e no *finally* chamamos os métodos responsáveis por destruir a instância.

Listagem 6. Função de execução da compressão

```
01 function TCompressaoPDF.Executar: Boolean;
02 const
03   Nparams = 16;
04 var
05   argv   : PPChar ;
06   code   : integer;
07   instance: Pointer;
08 begin
09   result := False;
10
11   setlength(argv, Nparams);
12   argv[0] := '-dSAFER';
13   argv[1] := '-dBATCH';
14   argv[2] := '-dNOPAUSE';
15   argv[3] := '-sDEVICE=pdfwrite';
16   argv[4] := '-dCompressFonts=true';
17   argv[5] := '-dPDFSETTINGS=/ebook';
18   argv[6] := PAnsiChar(AnsiString('-dDetectDuplicateImages='+
     DetectDuplicateImages));
19   argv[7] := '-dDownsampleColorImages=true \';
20   argv[8] := '-dDownsampleGrayImages=true \';
21   argv[9] := '-dDownsampleMonoImages=true \';
22   argv[10]:= PAnsiChar(AnsiString('-dColorImageResolution=' +
     IntToStr(ResolutionColor) + ' \'));
23   argv[11]:= PAnsiChar(AnsiString('-dGrayImageResolution=' +
      IntToStr(ResolutionGray)  + ' \'));
24   argv[12]:= PAnsiChar(AnsiString('-dMonoImageResolution=' +
      IntToStr(ResolutionMono)  + ' \'));
25   argv[13]:= PAnsiChar(AnsiString('-sOutputFile=' + Output));
26   argv[14]:= '-f';
27   argv[15]:= PAnsiChar(AnsiString(Input));
28
29   try
30     code := gsapi_new_instance(@instance, nil);
31     if code <> 0 then
32       raise Exception.Create('Erro ao instanciar. Código:'+ IntToStr(code));
33
34     code := gsapi_init_with_args(instance, Nparams, argv);
35     if code < 0 then
36       raise Exception.Create('Erro ao converter. Código:+ IntToStr(code));
37
38     result := True;
39   finally
40     gsapi_exit(instance);
41     gsapi_delete_instance(instance);
42   end;
43 end;
```

Para concluirmos, criaremos uma interface para instanciar e passar os valores desejados para um objeto do tipo TCompressaoPDF. A unit1 do projeto deve ser renomeada para uMain e o formulário renomeado para frmMain. O formulário deve ficar parecido com o da **Figura 1**, onde temos dois edits, um referente ao documento a ser comprimido e o outro ao local e nome do documento de saída. Para testar os níveis de compressão utilizaremos um CheckBox para determinar se iremos aplicar compressão sobre imagens duplicadas e um TrackBar com valor entre 0 a 300 para definirmos a resolução das imagens no arquivo. Deve ser declarada na seção *uses* a unit uCompressaoPDF. Nessa unit será implementado apenas um método que será no evento OnClick do botão Executar, cujo código pode ser visto na **Listagem 7**.

No evento OnClick do botão Executar foi declarada uma variável do tipo TCompressaoPDF, que é instanciado logo abaixo e tem suas propriedades preenchidas com os valores coletados no formulário. Para a propriedade DetectDuplicateImages é

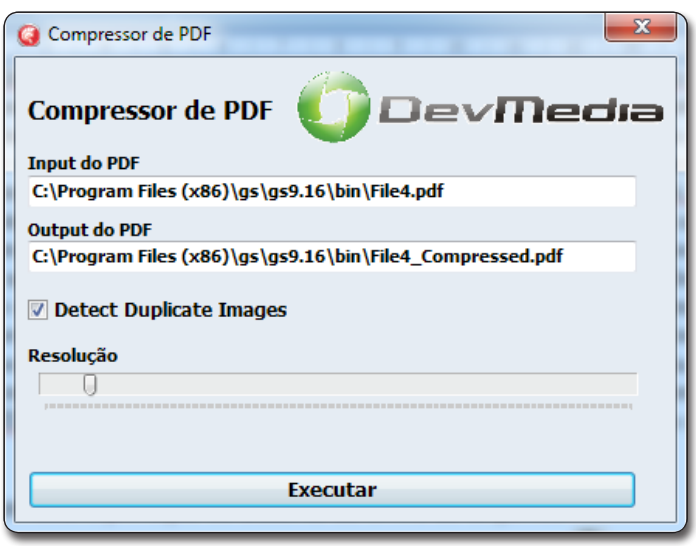


Figura 1. Formulário principal do sistema.

DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

efetuada uma validação: se o valor for *True* então é passado para a propriedade "*true*" como string e caso contrário é passado "false". Com isso concluímos a aplicação de compressão.

Listagem 7. Método do evento onclick do executar

```
01 procedure TfrmMain.BtnExecutarClick(Sender: TObject);
02 Var CompressaoPDF : TCompressaoPDF;
03 begin
04  CompressaoPDF := TCompressaoPDF.Create;
05  try
06   CompressaoPDF.Input        := edtInput.Text;
07   CompressaoPDF.Output       := edtOutput.Text;
08   CompressaoPDF.ResolutionColor := trckbrResolucao.Position;
09   CompressaoPDF.ResolutionGray  := trckbrResolucao.Position;
10   CompressaoPDF.ResolutionMono  := trckbrResolucao.Position;
11
12   if chkDetectDuplicateImages.Checked then
13    CompressaoPDF.DetectDuplicateImages := 'true'
14   else
15    CompressaoPDF.DetectDuplicateImages := 'false';
16
17   If CompressaoPDF.Executar Then
18    ShowMessage('Arquivo gerado com sucesso!')
19   else
20    ShowMessage('Falha ao gerar arquivo!');
21
22  finally
23   FreeAndNil(CompressaoPDF);
24  end;
25 end;
```

Para avaliação dos resultados vamos utilizar um arquivo que não está disponível na pasta do GhostScript, mas estará disponível juntamente ao código fonte deste artigo, que é *File4.pdf*, um arquivo com 811KB contendo apenas imagens duplicadas, excelente para nossos testes. Ao executar o sistema e gerar um PDF a partir dele, podemos acompanhar os resultados que são listados na **Tabela 3**.

Ao analisar os arquivos de saída veremos nitidamente que naqueles em que a resolução está muito baixa a perda de qualidade é percebida a ponto de em alguns casos não ser mais possível identificar a imagem. Os arquivos com valor de resolução até 100 ainda são arquivos de qualidade considerável, tendo uma boa nitidez e alta compressão. A partir dessa tabela podemos concluir que vale a pena utilizar a configuração para detectar imagens duplicadas, visto que o ganho na compressão é de mais que a metade em alguns casos, porém isso vai depender da quantidade de imagens repetidas no arquivo.

O GhostScript não se resume às funcionalidades que foram exploradas nesse artigo, aqui utilizamos apenas a parte do PostScript, que é a uma API responsável por trabalhar com PDF. O PostScript ainda tem uma infinidade de recursos e podemos programar e adicionar novas funcionalidades. Se analisarmos sua documentação, veremos que o número de parâmetros é enorme e para cada combinação desses argumentos temos novos resultados.

São diversas as possibilidades que podem ser exploradas utilizando a API do GhostScript. Por exemplo, ao extrair o texto de

Detect Duplicate Images	Resolução	Tamanho do arquivo gerado
Sim	300	445 KB
Não	300	885 KB
Sim	200	445 KB
Não	200	885 KB
Sim	150	235 KB
Não	150	493 KB
Sim	100	136 KB
Não	100	267 KB
Sim	50	62 KB
Não	50	109 KB
Sim	25	38 KB
Não	25	56 KB

Tabela 3. Comparação de compressão pelo aplicativo

um arquivo PDF podemos efetuar diversos procedimentos sobre os dados obtidos, desde simples validações para armazenamento em bases locais até mineração para extração de informações estratégicas baseadas em arquivos gerados em grande volume.

Para o cliente, a redução do espaço utilizado em disco reflete em economia com manutenção e aquisição de novos equipamentos, e em cenários onde esses arquivos são armazenados em uma base externa (com backup na nuvem, por exemplo),

Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

a diferença pode ser notada já nos primeiros períodos, com a redução do tráfego de dados e espaço alocado.

Quando os arquivos forem enviados aos clientes, a redução no tamanho tende a garantir uma experiência de uso mais fluída, uma vez que o download e abertura do arquivo se tornarão mais rápidos. Em cenários onde o cliente precisa, por exemplo, fazer download desses arquivos através de uma conexão 3G, de baixa velocidade, e com limite de tráfego reduzido, essa necessidade se torna ainda mais crítica.

Com base nas funcionalidades oferecidas pelo GhostScript, pode-se atender diversas demandas, bastando utilizar os parâmetros adequados. No site oficial da ferramenta, cujo endereço pode ser visto na seção **Links**, podem ser encontrados maiores detalhes sobre licença, documentação, bugs conhecidos, além de canais para interação com a comunidade que utiliza e contribui com o desenvolvimento da ferramenta.

Autor

Gutierry Antonio
gutierrydsn@hotmail.com

Entusiasta de Big Data e Data Warehouses, atua como Software Engineer e DBA MySQL. É graduado em Sistemas da Informação e Análise de Sistemas, e atualmente cursa especialização em Engenharia de Sistemas. Possui experiência em desenvolvimento com Delphi e Ruby on Rails com bancos de dados Firebird e MySQL, além de conhecimento de Java, C# e C++ com SQL Server e Oracle, bem como de AWS.

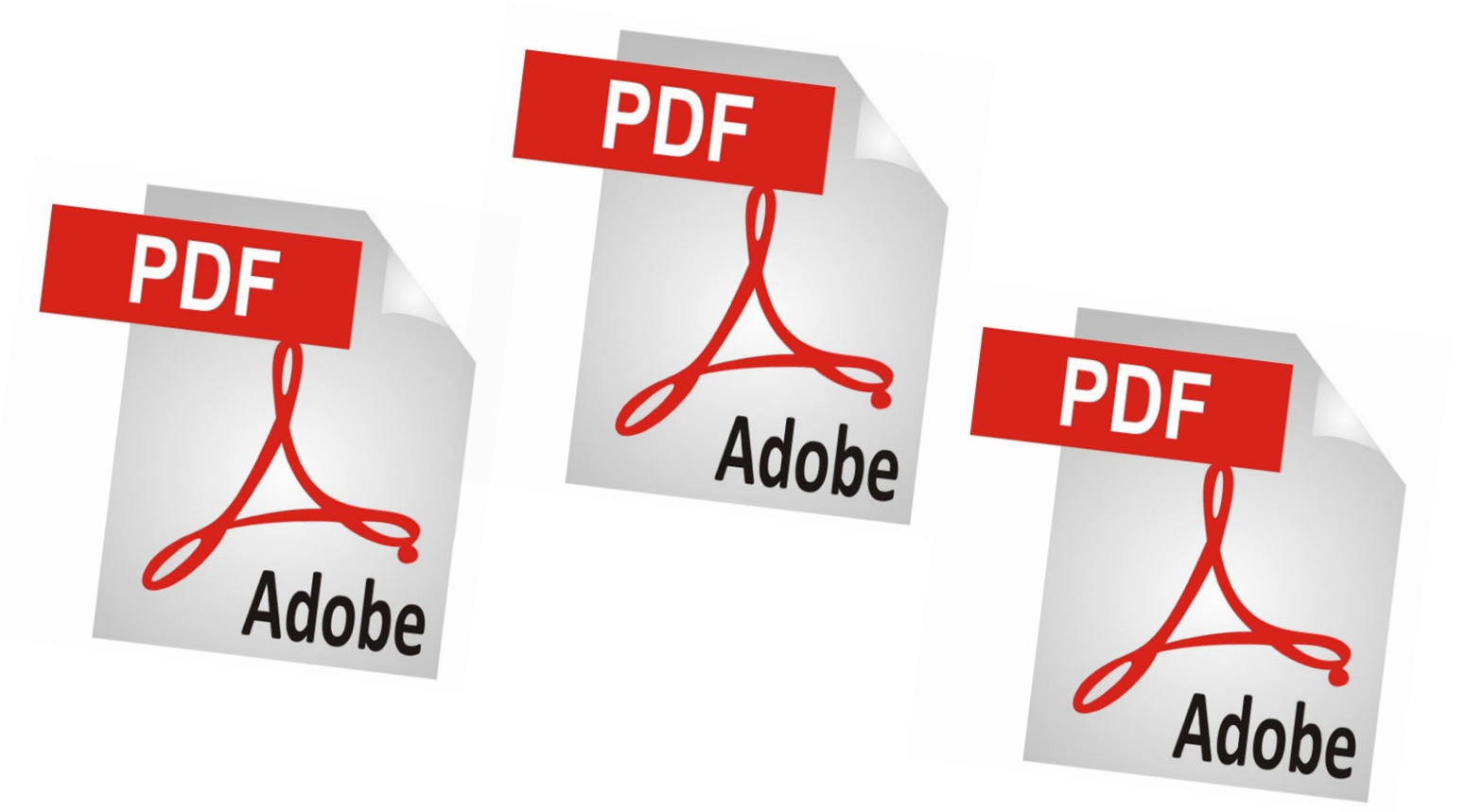
Links:

Site oficial do GhostScript
http://www.ghostscript.com/

Você gostou deste artigo?

Dê seu voto em ***www.devmedia.com.br/clubedelphi/feedback***

Ajude-nos a manter a qualidade da revista!



DEVMEDIA Copyright - Proibido copiar ou distribuir. Todos os direitos reservados para DevMedia

Guia HTML 5

Um verdadeiro manual de referência com tudo que você precisa sobre HTML!

DEVMEDIA

http://www.devmedia.com.br/guias/guia-html/3

Somos tão apaixonados por tecnologia que o nome da empresa

diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,

assim como você.

Estrutura

100% NACIONAL. Servidores de primeira linha, links de alta capacidade.

Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.

Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.

1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única.
Conheça!

Porta 80
WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce | Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486